Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем

Технология программирования

Подшиблов Александр Александрович

Разработка и реализация облачного сервиса для платформы Microsoft Azure для планирования повседневной учебной и научной деятельности студента

Бакалаврская работа

Научный руководитель:
ст. преп. Кузьменко В.Г.

Рецензент:
доцент, кандидат физ.-мат. наук Лебединский Д.М.

Санкт-Петербург

2016

SAINT-PETERSBURG STATE UNIVERSITY

Mathematical Software and Information Systems Administration

Technology of Programming

Alexander Podshiblov

Cloud service development and implementation for Microsoft Azure platform for students planning daily learning and research activities

Bachelor's Thesis

Scientific supervisor:
Senior Lecturer Kuzmenko Victor

Reviewer:
Assistant professor, Candidate of
Physical and Mathematical Sciences Lebedinski Dmitri

Saint-Petersburg

2016

Оглавление

1. Введение

Жизнь современного студента часто является очень насыщенной. И что естественно, большую часть его жизни занимает учебная и научная деятельность: посещение лекций и семинаров, выполнение домашних заданий, подготовка докладов и рефератов, написание курсовых работ. Также нужно не забыть о «контрольных» точках, днях, когда планируются промежуточные тесты, контрольные или коллоквиумы. А если к этому прибавить то, что многие студенты хотят быть активными помимо учебы сразу во многих областях жизни: заниматься любимым делом, интересно проводить свободное время, общаться с любимым человеком, поддерживать отношения с друзьями, то держать все в голове становится невозможно или, по крайней мере, очень проблематично.

Стремление успеть множество дел в ограниченное время иногда приводит к тому, что, погружаясь в какое-то одно дело или область, легко забыть о других. Возникают такие ситуации как пропущенные дедлайны, осознание того, что завтра контрольная работа или тест, а готовиться уже поздно. Автор данной работы и сам, будучи представителем студентов, сталкивался с такими ситуациями. Память нередко подводит студента, и это неудивительно, ведь ему приходится держать в голове довольно большой объем информации, который постоянно растет в процессе обучения. Возникает вопрос: как распланировать свою учебную деятельность заранее и не забыть об этом в последствие?

Естественным решением на первый взгляд является ведение блокнота или ежедневника с записями важной информации. То есть фиксировать информацию на бумажных носителях. Но такой подход имеет ряд недостатков и ограничений.

Во-первых, используя такой вид ежедневника, мы получаем серьезное ограничение в редактировании наших записей. Возможные зачеркивания, переписывания и стирания записей вносят хаос и занимают много времени.

Во-вторых, классификация, сортировка и поиск информации вносят еще больше неудобств. Кроме того, взяв в руки блокнот, мы не можем сразу понять, что же нас ожидает в ближайшее время и чем нам сейчас лучше заняться. И ростом активных задач время, затрачиваемое на это, растет более чем линейно. А если вспомнить, что разные дела имеют разную важность и полезность, то есть приоритет, то принятие решения на что лучше сейчас потратить свое время, начинает занимать значительное количество этого самого времени.

Поняв эти недостатки бумажных ежедневников, как средства планирования деятельности, логично обратить внимание на возможности персональных компьютеров, позволяющие выполнять все эти операции более эффективно. Они позволяют не только группировать дела, но и быстро сортировать их по какому-либо признаку, редактировать не только название дел, но и их атрибуты, указывать приоритет, выполнять быстрый поиск дел и распределять их по календарю.

Но у компьютерных органайзеров есть один существенный недостаток — их нужно устанавливать на каждый компьютер, за которым работает человек (дома, на работе, на ноутбуке…) и постоянно переносить данные о своих делах с одного компьютера на другой.

С развитием Интернета появилась возможность создать органайзер на одном сервере и хранить на нем данные большого количества пользователей, а доступ можно получить с любого устройства, подключенного к Интернету (компьютер, КПК, планшет, телефон…). И этот подход не имеет недостатков, которые были у предыдущих претендентов, и кажется наиболее оптимальным.

Кроме того, если у нас будет мобильный клиент для телефона, то получаем большую мобильность. Ведь телефон всегда с нами, его легко достать в любом месте, в отличие от ноутбука или ежедневника, и нет необходимости носить с собой что-то еще.

Имея в распоряжении такой инструмент планирования, студент сможет повысить эффективность своей учебной деятельности за счет концентрации на фактическом выполнении дел, а не траты своего времени на их постоянное повторяющееся обдумывание. Так же исключаются вероятность того, что он забудет важное или срочное дело. Но это требует дисциплинированности, т.к. нужно материализовывать всю информацию и постоянно ее актуализировать.

С учетом стиля жизни современного студента, ему необходим компактный и удобный инструмент планирования его учебной и научной деятельности, который будет всегда под рукой.

Тема данной работы была выбрана с учетом актуальности проблемы планирования студентами своей учебной деятельности. В работе рассматриваются популярные программные продукты, предназначенные для управления списком дел и событий с возможностью хранения данных на удаленных ресурсах с доступом посредством разнообразных типов устройств. Производится изучение современных облачных сервисов, их возможностей хранения, доступа и обработки данных. Результатом исследований является практическая разработка облачной базы данных, обрабатывающий ее мобильный сервис и связанное с ним мобильное приложение, позволяющее студенту вести учет и планирование учебной и научной деятельности.

2. Цель и постановка задачи

Целью данной работы является предоставить инструмент планирования повседневной учебной и научной деятельности студента путем реализации облачного сервиса для платформы Microsoft Azure и связанного с ним мобильного приложения.

Основными критериями, которым следует удовлетворять реализуемому приложению, являются:

Хранение данных в облачной инфраструктуре:

- Удобство внесения, просмотра и изменения информации о расписании и событиях
- Хранение данных в облачной структуре
- Аутентификация через какую-либо социальную сеть
- Синхронизация между устройствами

При разработке приложения основной акцент будет на соответствие и удовлетворение уникальных потребностей студентов, а также на удобство использования.

3. Обзор существующих продуктов

В данной главе проведен обзор наиболее популярных органайзеров для студентов. При написании главы использовались данные, предоставляемые авторами рассматриваемых программных продуктов, а также сведения, полученные аналитическим путем.

3.1. iStodo

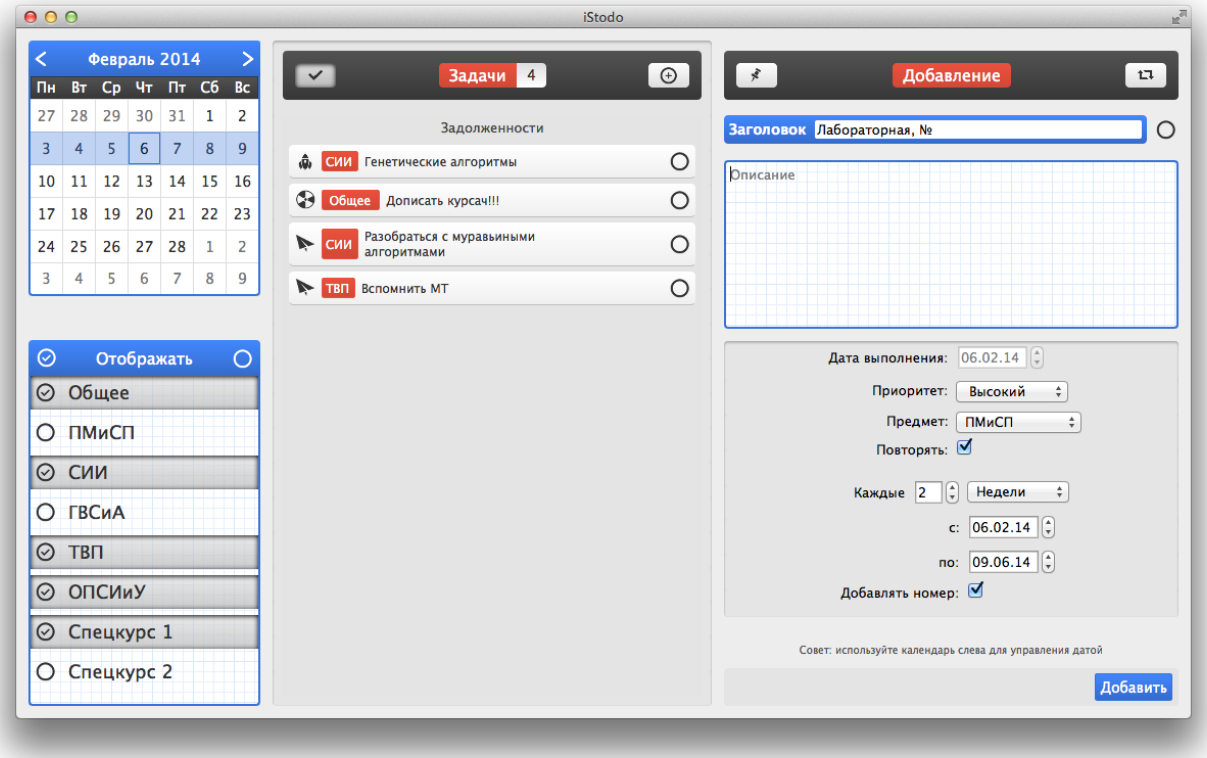

Рис. 1. Интерфейс приложения IStodo

Предлагаемое решение помогает планировать и распределять нагрузку на семестр вперед, расставлять задачам приоритеты, контролировать сроки выполнения. [1] Вы можете настроить расписание под себя, например, убрать из сетки, параллельно идущие, пары второй подгруппы, или наоборот, добавить

консультации преподавателя. Вся информация за неделю представлена в наглядном виде, на одном экране — использовать разрозненные инструменты больше нет необходимости.

Ключевые особенности продукта:

- Работа с сериями заданий, фильтрация, приоритеты, сроки выполнения
- Визуальный редактор расписания
- Поддержка расписания из отличающихся (четные/нечетные) недель
- Поддержка основных ОС: Windows, OS X, Linux
- Возможность экспорта расписания и задач в формате iCal
- Возможность создания резервных копий: позволяет перенести всю информацию на другой компьютер, поделиться с одногруппниками
- Адаптирован к экранам нетбуков(1024*600 px)

Главным недостатком можно отметить хранение данных локально, что ограничивает мобильность и гибкость использования, а также нивелирует мультиплатформенность программного продукта.

Отсутствует возможность добавить домашнее задание на конкретную пару, расписание существует только как расписание. Задачи добавлять можно, но все задания идут в одном разделе «Задачи» и из свойств имеют только приоритет, дедлайн и предмет, по которому стоит задача. Отсутствуют типы задач.

Данное приложение можно рассматривать как продвинутое расписание и с возможностью добавлять задачи, но не средство для планирования учебной деятельности.

3.2. Timetable

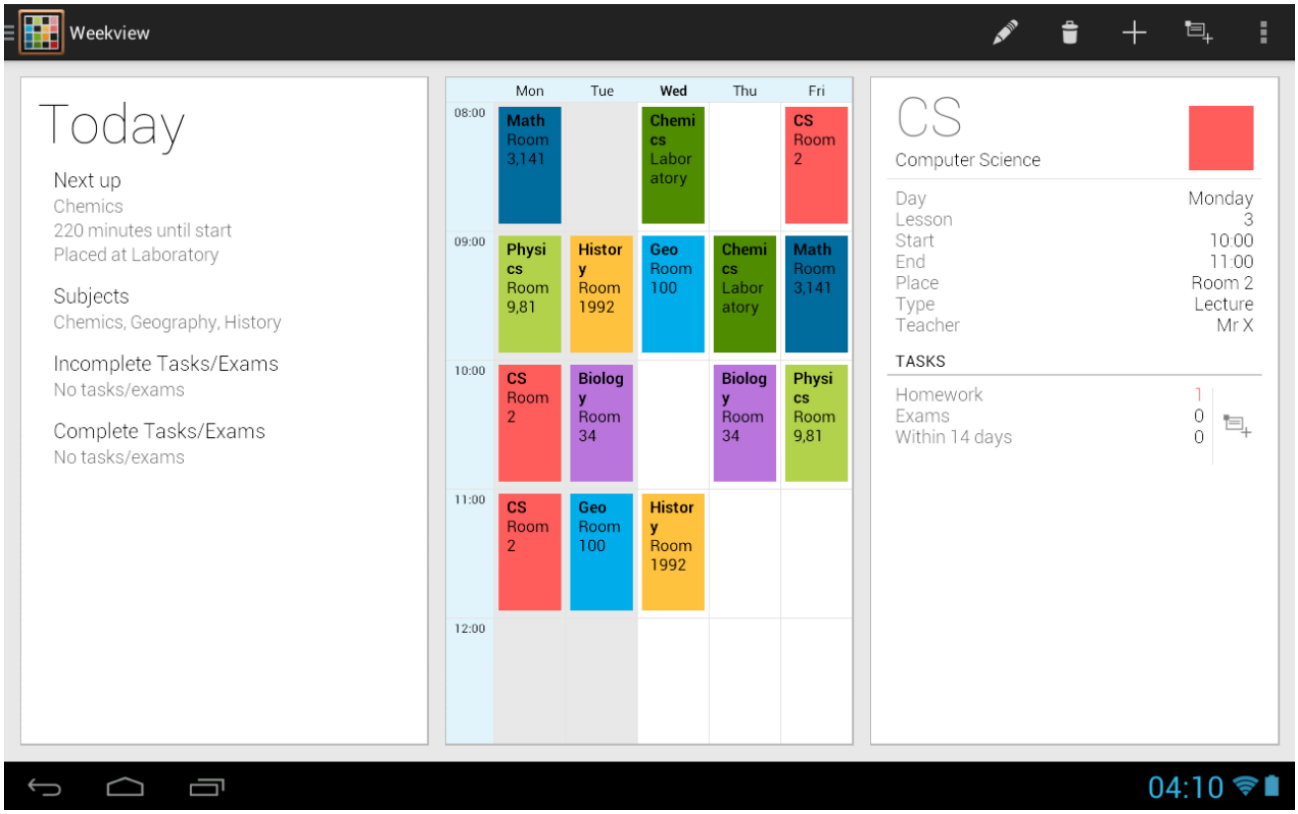

Рис. 2. Интерфейс приложения Timetable

Timetable — приложение, предназначенное для управления вашей школьной или университетской жизнью. [2] Вы можете сохранять ваше расписание и все задания, начиная домашними заданиями и заканчивая экзаменами. Все это нужно будет ввести только один раз, так как Timetable синхронизируется между всеми вашими Android устройствами. Приложение может само переводить ваше устройство в беззвучный режим во время занятий.

Краткий обзор

- Синхронизация информации между устройствами
- Сохранение занятий и заданий
- Просмотр расписания в виде списка и в виде таблицы
- Поиск в расписании и в заданиях

- Возможно включение второй, третьей и четвертой учебной недели
- Уведомления о занятиях и заданиях на завтра
- Автоматический беззвучный режим во время занятий

Из недостатков, исходя из целей, которые мы преследуем, можно отметить недостаточную специализацию именно для студентов. Из типизированных заданий есть только экзамены. Кроме того нет приоритетов задач. Несмотря на возможность синхронизации, приложение хранит данные локально, из-за чего появляется риск их утраты.

3.3. Выводы и идеи

Итогом работы над данным обзором является набор выводов и идей по необходимой функциональности, которая сделала бы реализуемое приложение полезным и удобным для использования, а также учитывала специфику учебной деятельности студентов.

Далее приведены выводы и идеи по функционалу будущего приложения, по средствам чего оно бы превосходило аналоги:

- Облачное хранение данных
- Возможность добавления домашнего задания на конкретную пару
- Различные типы задач с уникальными свойствами и инструментами
- Приоритетность задач
- Указание предполагаемого времени выполнения для краткосрочных задач
- Пометка задач как выполненных
- Своевременное информирование об активных задачах с учетом их предполагаемого времени выполнения
- Напоминания о необходимости выполнения долгосрочных задач с учетом общей загруженности студента

4. Обзор платформы Micrisoft Azure

Как уже хорошо известно, облако (cloud) – широко используемая метафора для изображения сервисов, предоставляемых через Web.[3]

Облачные вычисления – модель вычислений, основанная на динамически масштабируемых (scalable) виртуализованных (virtual) ресурсах - данных, приложениях, Web-сайтах, виртуальных машинах, ОС и др., - которые доступны и используются как сервисы через Интернет и реализуются с помощью мощных центров обработки данных (data centers).

Пользователям доступны "облака" (общедоступные, частные или "облака сообществ"), предоставляемые различными компаниями и организациями для использования мощных вычислительных ресурсов, которых нет у индивидуального пользователя.

Наиболее популярная "облачная" платформа – Microsoft Windows Azure (облачная ОС) и Microsoft Azure Services Platform (реализованная на основе Microsoft.NET).

4.1. Развитие концепций облачных вычислений: модели обслуживания

Какого рода услуги предоставляются в облаке, и какие модели обслуживания используются? Рассмотрим классификацию этих моделей, в которой за недавние годы появились новые элементы.

Инфраструктура как сервис (Infrastructure as a Service - IaaS) – модель обслуживания клиентов облака, при которой провайдер облака предлагает реальные или виртуальные машины и их ресурсы: образы дисков, виртуальные локальные сети и др. Виртуальная инфраструктура (за небольшую арендную плату или вовсе бесплатно, причем без необходимости делать какие-либо инсталляции на своих компьютерах) – это и есть самое ценное в облачных

вычислениях, одна из основных причин, почему столь большое число клиентов начало использовать облако.

Платформа как сервис (Platform as a Service - PaaS) – модель обслуживания клиентов облака, при которой провайдер облака предлагает клиентам целую компьютерную платформу: операционную систему, окружение для выполнение программ на языках программирования, базу данных и Web-сервер. К этому классу моделей относится Microsoft Azure.

Программное обеспечение как сервис (Software as a Service - SaaS) – модель обслуживания клиентов облака, при которой провайдер облака инсталлирует в облаке прикладные программы, которые используются клиентами облака. Яркий пример – Google Cloud Apps, облачные решения фирмы Google, полезные приложения, которые легко интегрировать в браузеры клиентов.

Сеть как сервис (Network as a Service – NaaS) – относительно новый вид облачных услуг, при котором провайдер облака предлагает клиентам сетевые услуги: транспорт по сети, виртуальные частные сети (VPN) и др. Пример – облачный вариант электронной почты, который бесплатно предлагается в настоящее время многими фирмами.

Microsoft Azure – облачная Интернет-платформа, разработанная фирмой Microsoft (по существу, операционная система и набор инструментов "в облаке"). Microsoft Azure обеспечивает хранение, использование, модификацию данных и запуск программ на компьютерах центров обработки данных Microsoft. Никакого программного обеспечения, кроме веб-браузера, на пользовательских компьютерах не требуется.[4]

Microsoft Azure полностью реализует две облачные модели — платформы как сервиса (Platform as a Service, PaaS) и инфраструктуры как сервиса (Infrastructure as a Service, IaaS). Работоспособность платформы Windows Azure обеспечивает сеть глобальных дата-центров Microsoft.

Основные особенности данной модели:

- оплата только потребленных ресурсов
- общая, многопоточная структура вычислений
- абстракция от инфраструктуры

В основе работы Microsoft Azure лежит запуск виртуальной машины для каждого экземпляра приложения[5]. Разработчик определяет необходимый объем для хранения данных и требуемые вычислительные мощности (количество виртуальных машин), после чего платформа предоставляет соответствующие ресурсы. Когда первоначальные потребности в ресурсах изменяются, в соответствии с новым запросом заказчика платформа выделяет под приложение дополнительные или сокращает неиспользуемые ресурсы дата-центра.

Microsoft Azure как PaaS обеспечит не только все базовые функции операционной системы, но и дополнительные: выделение ресурсов по требованию для неограниченного масштабирования, автоматическую синхронную репликацию данных для повышения отказоустойчивости, обработку отказов инфраструктуры для обеспечения постоянной доступности и многое другое.

Microsoft Azure также реализует другой тип сервиса — инфраструктуру как сервис. Модель предоставления инфраструктуры (аппаратных ресурсов) реализует возможность аренды таких ресурсов, как серверы, устройства хранения данных и сетевое оборудование. Управление всей инфраструктурой осуществляется поставщиком, потребитель управляет только операционной системой и установленными приложениями. Такие сервисы оплачиваются по фактическому использованию и позволяют увеличивать или уменьшать объем инфраструктуры через специальный портал, предоставляемый поставщиками. В данной сервисной модели могут быть запущены практически любые приложения, установленные на стандартные образы ОС.

4.2. Мобильные сервисы Windows Azure

В рамках данной работы мы будем использовать облако Microsoft Azure и его компоненту Azure Mobile Apps.

Мобильный сервис – это облачный сервис, доступный для использования с мобильного устройства (например, смартфона)[6]. Мобильные сервисы Windows Azure позволяют быстро создавать привлекательные межплатформенные и собственные приложения для iOS, Android, Windows или Mac. Они могут сохранять данные приложений в облаке или локально, выполнять авторизацию пользователей, отправлять push-уведомления. Кроме того, службы дают возможность добавить вашу пользовательскую логику доступа к базе данных в приложения C# или Node.js.

Авторизация пользователей легко выполняется через Active Directory. Благодаря этим службам доступно безопасное подключение к таким локальным ресурсам, как SAP, Oracle, SQL Server и SharePoint. Возможно также создание приложений корпоративного уровня для своих сотрудников с использованием комплексных платформ наподобие Xamarin и PhoneGap.

С помощью мобильных служб возможно создание надежных приложений, способных работать даже при возникновении проблем в сети. Это необходимо для того, чтобы пользователи могли создавать и изменять данные, работая в автономном режиме. Повышение скорости отклика приложения за счет локального кэширования серверных данных на устройстве. Мобильные приложения позволяют легко добиться бесперебойной синхронизации данных всех приложений iOS, Android и Windows.

Центры уведомлений — это мощный масштабируемый механизм рассылки мобильных push-уведомлений. Он способен с невероятной скоростью рассылать миллионы push-уведомлений на устройства iOS, Android, Windows или Nokia X. Концентраторы уведомлений можно легко связать с любой

существующей серверной частью приложения, независимо от того, где размещается эта часть: локально или в Azure.

Azure предлагает простой способ настройки встроенного автомасштабирования в соответствии с потребностями приложения для мобильных служб и концентраторов уведомлений. Автоматическое масштабирование позволяет развертывать или свертывать ресурсы в зависимости от их фактического использования и оплачивать только то, что вы используете. Открытый доступ к глобальной сети управляемых центров обработки данных Майкрософт позволяет предоставлять данные пользователям в любом месте по всему миру.

5. Спецификация приложения

Приложение Student Activity Manager основано на платформе Azure App Service, которая объединяет широкий набор возможностей для работы через Интернет, с мобильными устройствами и сценариев интеграции. В частности на ее компоненте Mobile Apps. Доступ к платформе Azure осуществляется по DreamSpark подписке, предоставляющей студентам и аспирантам ограниченный бесплатный доступ к данной платформе.

Контроль версий осуществляется посредством системы контроля версий Git с публикацией кода в публичный репозиторий на GitHub.

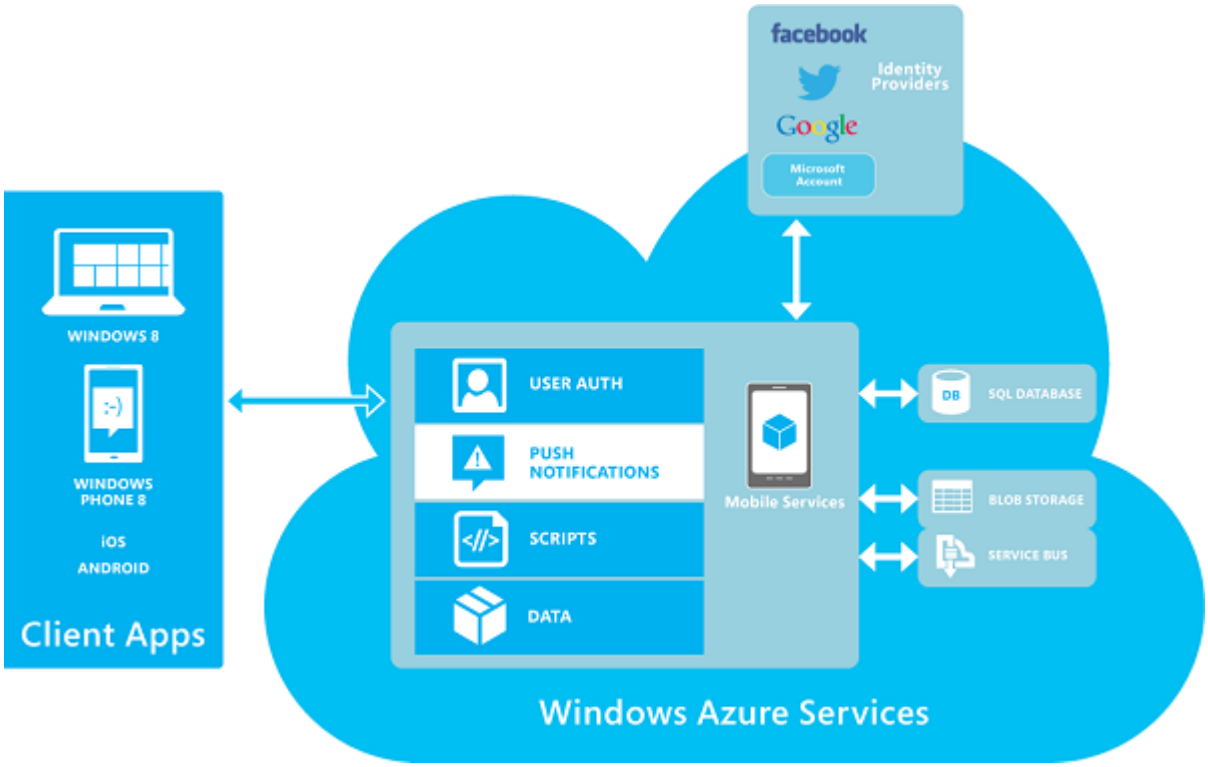

Рис. 3. Схема взаимодействия компонентов в мобильном сервисе Azure

5.1 Серверная часть

Серверная часть состоит из службы приложений Azure, сервера базы данных и базы данных SQL Azure, управляемых посредством портала Azure.

Язык серверной части – Node.js. Файлы сценариев серверной части редактируются с помощью инструмента Visual Studio Team Services.

Осуществляется авторизация и аутентификация через учетную запись Google. Ограничивается доступ к чужим данным в таблицах базы данных путем редактирования серверных Javascript сценариев.

5.2 Клиентское приложение

Клиентское приложение разрабатывается для платформы Android в среде разработки Android Studio с использованием инструментов Android SDK и пакета SDK для мобильных приложений Android в Azure.

Приложение должно иметь локальное хранилище и при необходимости синхронизировать его с сервером. Позволять хранить данные авторизации во избежание повторной ручной авторизации. По истечении срока действия авторизационных данных, автоматически запускать повторную авторизацию.

Далее будут описаны основные модули клиентского приложения и их функции.

5.2.1 Главное модуль

- Инициализация объекта мобильного клиента
- Авторизация пользователя
- Кэширование и загрузка токена авторизации
- Проверка регистрации пользователя
- Запуск модуля регистрации пользователя
- Кэширование и загрузка регистрационных данных пользователя
- Запуск модуля расписания

5.2.2 Модуль регистрации пользователя

- Загрузка с сервера списка существующих учебных заведений
- Загрузка с сервера списка существующих факультетов, в зависимости от выбранного учебного заведения
- Сбор пользовательских данных и регистрация пользователя

5.2.3 Модуль расписания

- Инициализация локального хранилища
- Синхронизация локального хранилища с сервером
- Отображение расписания
- Запуск модуля добавления нового элемента в расписание
- Запуск модуля редактирования элемента расписания
- Удаление элемента расписания
- Удаление связанных задач при удалении элемента расписания
- Запуск модуля задач для элемента расписания
- Запуск модуля всех задач
- Запуск модуля краткосрочных задач
- Запуск модуля долговременных задач

5.2.4 Модуль добавления нового элемента расписания

- Сбор информации о новом элементе расписания
- Запуск модуля добавления нового элемента времени
- Удаление элемента времени
- Добавление элемента в расписание

5.2.5 Модуль редактирование элемента расписания

- Загрузка информации о редактируемом элементе расписания
- Сбор отредактированной информации об элементе расписания

- Запуск модуля добавления нового элемента времени
- Удаление элемента времени
- Обновление элемента в расписании

5.2.6 Модуль добавления нового элемента времени

- Сбор информации о новом элементе времени
- Добавление нового элемента времени

5.2.7 Модуль задач для элемента расписания

- Отображение задач, назначенных на текущий элемент расписания
- Пометка задач как выполненных
- Запуск модуля добавления новой задачи
- Удаление задачи

5.2.8 Модуль всех задач

- Отображение задач, назначенных на текущий элемент расписания
- Пометка задач как выполненных
- Удаление задачи

5.2.9 Модуль новой задачи

- Сбор информации о новой задаче
- Добавление новой задачи

5.2.10 Модуль краткосрочных задач

- Отображения списка краткосрочных задач
- Пометка задач как выполненных
- Запуск модуля новой задачи

5.2.11 Модуль долговременных задач

- Отображение древовидного списка долговременных задач
- Пометка задач как выполненных
- Отображение прогресса выполнения задачи
- Удаление задачи
- Запуск модуля новой задачи
- Запуск модуля создания нового уведомления

5.2.10 Модуль нового уведомлений

- Сбор информации о времени уведомления
- Регистрация уведомления в системе
- Отправка уведомлений о необходимости выполнения задач

5.2.11 Модуль демонстрации уведомлений

- Прием сообщений от системы о необходимости демонстрации уведомления
- Демонстрация уведомления

На следующей диаграмме можно увидеть потоки данных между основными компонентами системы.

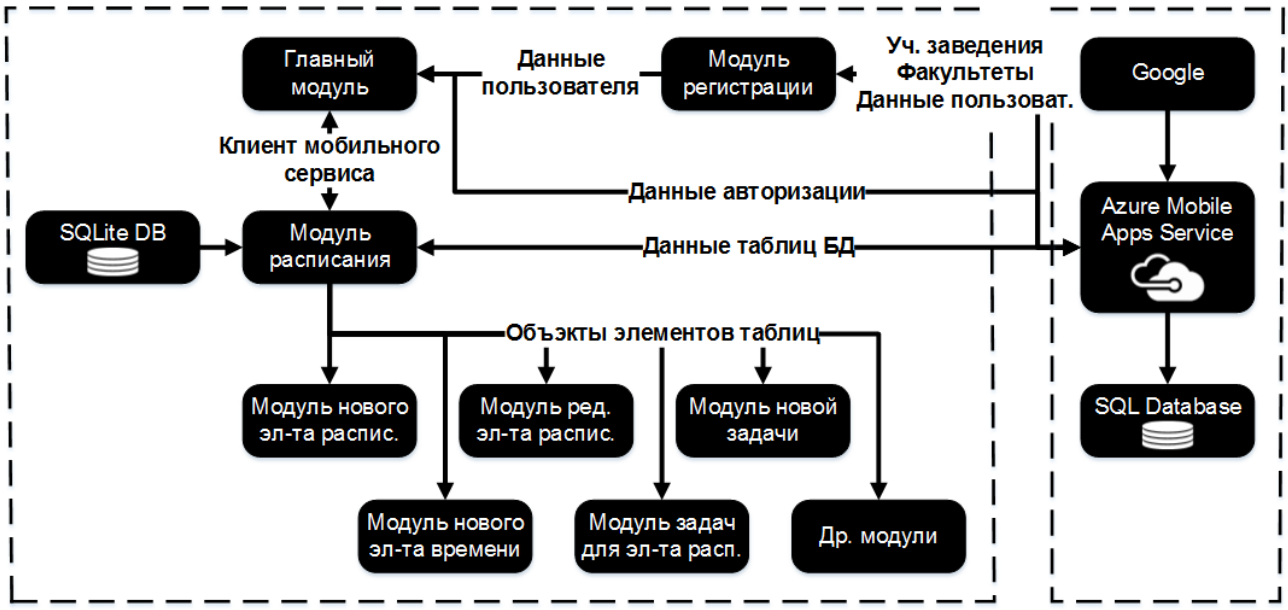

Рис. 4. Диаграмма потоков данных между основными компонентами системы

6. Описание реализации

6.1. Авторизация пользователей по учетной записи Google с кэшированием токена авторизации

Служба приложений использует федеративное удостоверение, когда сторонний поставщик удостоверений сохраняет учетные записи и проверяет подлинность пользователей, а приложение использует это удостоверение вместо своего. Служба приложений поддерживает пять поставщиков удостоверений по умолчанию: Azure Active Directory, Facebook, Google, учетная запись Майкрософт и Twitter.[7]

6.1.1 Серверная часть

Для добавления авторизации пользователя с помощью поддерживаемого поставщика удостоверений (в данном случае это Google) в мобильное приложение Azure необходимо зарегистрировать его на сайте используемого поставщика удостоверений.

На сайте Google APIs создаем проект нашего приложения. Затем в разделе Google+ API создаем новый «OAuth 2.0 client ID» (Идентификатор клиента OAuth 2.0) типа Web application (Веб приложение). В качестве Authorized JavaScript Origins (Авторизованные источники JavaScript) используем URL нашего приложения, который можно увидеть, перейдя в портал Azure к своему приложению. Authorized redirect URIs (Авторизованный URI перенаправления) строится путем дополнения URL-адреса приложения путем: /.auth/login/google/callback. Далее мы получим строки Client ID и Client Secret, которые понадобятся нам в будущем для настройки серверной части нашего приложения.

Для настройки серверной части приложения переходим в портал Azure. В параметрах нашего приложения, в разделе аутентификации или авторизации

активируем авторизацию через Google, указываем значения идентификатора и секретного кода приложения, полученные ранее.

6.1.2 Клиентская часть

Определяем метод authenticate, который запускается при старте приложения до всякого запроса к базе данных. На вход он принимает логический параметр isForcibly, который управляет режимом работы, при котором игнорируется кэшированный токен авторизации (кэширование токена описано далее).

Для получения объекта типа ListenableFuture<MobileServiceUser> используем метод login(MobileServiceAuthenticationProvider.Google) класса MobileServiceClient (описание в разделе 2.1.2), который запускает процесс аутентификации с использованием указанного поставщика проверки подлинности (в нашем случае Google).

Добавляем обратный вызов (объект типа FutureCallback<MobileServiceUser>) для обработки успешного и провального выполнения авторизации путем переопределения методов onFailure и onSuccess интерфейса FutureCallback.

При неудачной авторизации показываем сообщение об обязательной авторизации. При успешном же прохождении авторизации кэшируем токен авторизации и запускаем проверку регистрации пользователя (раздел 2.2).

6.1.2.1 Кэширование токена авторизации

Обращаться как к поставщику удостоверений, так и к серверной службе Azure каждый раз, когда приложение запускается неэффективно, поэтому было решено кэшировать токен авторизации, который возвратила служба Azure.

Реализуем метод cacheUserToken(MobileServiceUser user), принимаемый этим методом объект пользователя можно получить вызовом метода getCurrentUser() у объекта класса MobileServiceClient (описание в разделе 2.1.2).

Хранить данные авторизации мы будем, используя интерфейс SharedPreferences. Получаем объект, используя метод getSharedPreferences(FILE_NAME, Context.MODE_PRIVATE). Константа MODE_PRIVATE используется для настройки доступа и означает, что после сохранения, данные будут видны только этому приложению.

Используя Editor, полученный вызовом метода edit, записываем идентификатор пользователя и токен авторизации, которые получаем из объекта класса MobileServiceUser методами getUserId и getAuthenticationToken соответственно.

6.1.2.2 Загрузка кэшированного токена авторизации

В начале авторизации делается попытка загрузить сохраненный токен авторизации. Реализуем метод loadUserTokenCache(MobileServiceClient client), возвращающий true при успешной загрузке токена.

Используя метод getString объекта SharedPreferences, полученному аналогично предыдущему разделу, пытаемся получить кэшированные данные. В случае неудачи возвращаем false, иначе создаем объект MobileServiceUser, в конструктор которого передаем идентификатор пользователя, и вызовом его метода setAuthenticationToken устанавливаем токен авторизации. Далее методом setCurrentUser объекта класса MobileServiceClient устанавливаем текущего пользователя и возвращаем истину.

6.1.2.3 Обновление токена авторизации

В случае, если срок действия токена авторизации истек, при обращении к серверу мы получим исключение с информацией «{'code': 401}». В этом случае в обработчике исключения вызываем метод authenticate с параметром true, что вызовет запуск авторизации с игнорированием кэшированного токена.

6.2. Регистрация пользователей по университету и факультету

Мы будет хранить некоторую информацию о пользователе (университет и факультет), для возможного использования в будущем (например, если мы захотим добавить поддержку публичных расписаний, то будем показывать расписания пользователей того же факультета).

6.2.1 Работа с таблицами базы данных SQL Azure

SQL Azure - это облачный сервис от корпорации Microsoft, предоставляющий возможности по хранению данных. Azure SQL Databases основан на Microsoft SQL Server, но предоставляет только подмножество типов данных. Поддерживаются основные типы: строковый, числовой, дата и логический.

6.2.1.1 Серверная часть

Серверный пакет SDK azure-mobile-apps для Node.js предоставляет механизмы доступа через веб-интерфейсы к таблицам, хранящимся в базе данных SQL Azure. Пакетом предоставляются пять операций. [8]

GET /tables/_tablename_	Получает все записи в таблице
GET	Получает определенную запись из

/tables/_tablename_/:id	таблицы
POST /tables/_tablename_	Создает новую запись в таблице
PATCH /tables/_tablename_/:id	Обновляет существующую запись в таблице
DELETE /tables/_tablename_/:id	Удаляет запись из таблицы

Определять таблицы перед использованием можно, размещая их определения в отдельных файлах JavaScript в каталоге tables (рекомендуется) и затем использовать метод tables.import(), чтобы импортировать таблицы. Также имеется возможность управлять таблицами, используя инструмент "Easy Tables" на портале Azure.

Таблицы можно определять с помощью статических схем (когда разработчик определяет столбцы в схеме) или динамических схем (когда схема определяется пакетом SDK в зависимости от поступающих запросов).

Разработчик может управлять некоторыми деталями работы WebAPI путем редактирования функций, обрабатывающих запросы, на JavaScript (например, фильтровать результаты в зависимости от id пользователя). Портал Azure позволяет редактировать файлы сценариев серверной части Node.js в Visual Studio Team Services без скачивания проекта на локальный компьютер.[9] Этим инструментом и пользовался автор данной работы.

6.2.1.2 Клиентская часть

Для доступа к данным из таблиц SQL Azure нужно определить клиентские классы данных, которые соответствуют таблицам в серверной части мобильного приложения. [10] Например, если таблица содержит столбцы:

- id
- text
- complete

Соответствующий класс будет выглядеть следующим образом:

```
public class ToDoItem {
    private String id;
    private String text;
    private Boolean complete;
}
```

Для доступа к серверной части мобильного приложения необходим объект MobileServiceClient.

```
MobileServiceClient mClient = new MobileServiceClient(
            "MobileAppUrl",
            this)
```

Этот код размещается в методе onCreate класса Activity, указанного в файле AndroidManifest.xml в качестве действия MAIN и категории LAUNCHER.

Для доступа к таблице сначала создается объект MobileServiceTable, вызвав метод getTable объекта MobileServiceClient. Этот объект предоставляет методы execute, insert, update и delete для работы с таблицей в базе данных. Важно вызывать эти методы в фоновом потоке во избежание зависания пользовательского интерфейса.

6.2.2 Детали конкретной реализации

Для хранения базы учебных заведений используется таблица со столбцами id и name хранящими уникальный идентификатор учебного заведения и его название соответственно. Для факультетов используется таблица со столбцами id, name и edinstid, последний хранит идентификатор учебного заведения, которому принадлежит факультет. Список пользователей хранится в таблице со столбцами id и facultyid, в первом хранится идентификатор полученный при авторизации через Google, во втором идентификатор факультета, указанного при регистрации.

После успешной авторизации пользователя и кэширования его токена авторизации (либо загрузки кэшированного токена) запускается метод checkUser, проверяющее наличие и загружающий кэшированные регистрационные данные пользователя.

В случае отсутствия кэшированных регистрационных данных пользователя происходит запуск экрана с формой регистрации. Происходит запрос к базе данных и загружается список существующих учебных заведений. Для ввода имени учебного заведения используется объект типа AutoCompleteTextView со стандартным адаптером данных типа ArrayAdapter, которому в качестве набора данных передается полученный ранее список. Таким образом, пользователь, начав печатать имя своего учебного заведения, получает выпадающий список уже существующих заведений. Он может выбрать учебное заведение из списка, либо ввести свое.

После нажатия на кнопку подтверждения проверяется корректность данных и если пользователь ввел новое имя учебного заведения, отправляется запрос на вставку нового элемента в таблицу или загружается список факультетов, существующих в базе в рамках выбранного учебного заведения. Аналогичным образом инициализируется поле для ввода/выбора факультета.

После подтверждения введенных данных и проверки их корректности, в случае ввода нового имени факультета, выполняется запрос на вставку нового элемента в таблицу факультетов. Далее происходит вставка нового элемента в таблицу пользователей и кэширования регистрационных данных. Кэширования регистрационных данных пользователя происходит аналогичным кэшированию токена авторизации способом.

6.3 Автономная работа и синхронизация

Автономная синхронизация позволяет конечным пользователям взаимодействовать с мобильным приложением (просматривать, добавлять или изменять данные) даже при отсутствии подключения к сети. Изменения хранятся в локальной базе данных; после подключения устройства к сети эти изменения синхронизируются с удаленным внутренним сервером.[13]

При включенной синхронизации автономных данных операции осуществляются с помощью таблицы синхронизации (с помощью интерфейса IMobileServiceSyncTable), которая является частью базы данных SQLight на устройстве.

Для принудительной отправки изменений на устройстве в мобильные службы Azure и получения изменений из мобильных служб используется контекст синхронизации (MobileServiceClient.SyncContext), который инициализируется с помощью локальной базы данных.

Контекст синхронизации связан с объектом мобильного клиента (MobileServiceClient) и отслеживает изменения, внесенные таблицей синхронизации. Контекст синхронизации обеспечивает очередь операций, содержащую упорядоченный список операций CUD (Create, Update, Delete — создание, обновление, удаление), которые позднее будут отправлены на сервер.

Локальное хранилище связано с контекстом синхронизации посредством метода инициализации MobileServiceSyncContext. Initialize.

6.3.1 Инициализация локального хранилища

Реализуется метод initLocalStore(), инициализирующий локальное хранилище и контекст синхронизации.

Вначале, используя описанный выше метод мобильного клиента, получаем контекст синхронизации. Его методом isInitialized проверяем его инициализированнность и в случае возврата значения false приступаем к инициализации локального хранилища.

Инициализируем новый объект класса SQLiteLocalStore. Для добавления описания таблиц в этот объект, необходимо для каждой таблицы создать объект класса Map<String, ColumnDataType>, описывающий структуру таблицы, и наполнить его информацией о столбцах таблицы. Затем методом локального хранилища defineTable добавить описание таблицы в локальное хранилище. После добавления описаний всех таблиц, используемых для автономной синхронизации, методом initialize контекста синхронизации инициализируем его, передав в метод объект нашего локального хранилища.

6.3.2 Синхронизация

Реализуем метод refreshFromMobileServiceTableSyncTable, который будет синхронизировать данные, используя таблицы синхронизации.

Вначале вызывается метод sync, который мы реализуем для выполнения попытки синхронизации с сервером. В нем мы получаем контекст синхронизации и вызываем его метод push для отправки изменений на сервер. Затем для каждой таблицы синхронизации вызываем ее метод pull, загружающий изменения сервера.

Тут хотелось бы сделать небольшое отступление. Автор данной работы при тестировании приложения столкнулся со следующей проблемой: при выполнении операции pull ловилось исключение с сообщением

«…table.sync.localstore.MobileServiceLocalStoreException: android.database.sqlite.SQLiteException: near ",": syntax error: , while compiling: INSERT OR REPLACE INTO …». Синтаксическая ошибка в SQL запросе к локальной базе данных SQLite, в то время как явно мы этот запрос не писали, а использовали методы классов, предоставленных Azure SDK. Дело оказалось в следующем: до версии 3.7.11 SQLite не поддерживает множественную вставку строк, а эта версия появилась только в Android API 16 (Android 4.1), автор же данной работы использовал устройство на платформе Android 4.0.3.

Если отсутствует подключение к интернету то, поймав соответствующее исключение, просто продолжаем работу, данные будут сохраняться в локальном хранилище и синхронизироваться при наличии подключения во время следующего вызова метода синхронизации.

В случае возникновения исключения, говорящего о коде 401, мы понимаем, что истек срок действия токена авторизации и выполняем вызов метода авторизации с параметром true (форсированный режим, игнорирующий кэшированный токен).

6.4 Управление расписанием

Расписание пользователя хранится в виде элементов расписания в таблице со столбцами id, userId, title, classroom, day, timeItemId, хранящими соответственно: идентификатор элемента расписания, идентификатор пользователя, заголовок элемента, аудиторию, день недели и идентификатор элемента времени.

Элемент времени представляет собой сущность, хранящую информацию о временном интервале элемента расписания. Введен для оптимизации базы данных, а также для того, чтобы пользователю не приходилось каждый раз вводить временные рамки для сходных по времени элементов расписания (например, пользователь может создать элемент времени «Первая пара»,

хранящий временной интервал 9:30 – 11:05, и выбирать его при создании элемента расписания первой пары для разных дней недели).

Временные элементы хранятся в таблице со столбцами: id, userId, name, sh, sm, fh, fm. Назначение первых трех столбцов аналогично столбцам таблицы элементов расписания. Столбцы sh, sm, fh, fm отвечают за хранение временного интервала, часы и минуты начала и часы и минуты конца интервала соответственно.

6.4.1 На стороне клиентского приложения

При переходе к расписанию в верхней части экрана отображается элемент пользовательского интерфейса типа Spinner, позволяющий фильтровать отображаемые элементы расписания по дням недели, что сделано для экономии пространства экрана и удобства навигации.

Элементы расписания представляются посредством элемента ListView, которому назначен переопределенный адаптер данных. Переопределение адаптера данных, необходимо для изменения представления по умолчанию (по умолчанию отображается строка, полученная от объекта методом toString), а также поддержки фильтрации элементов по дню недели и сортировки по времени. На изменение выбранного элемента в Spinner’е выбора дня недели назначен слушатель, который запускает метод фильтрации у адаптера данных.

В элементе ListView отображается информация о времени начала и конца элемента расписания, а также его заголовок. Изменение представления строк ListView происходит путем добавления нового xml файла макета и наполнения его в методе getView нашего адаптера данных.

Также элементу ListView, отображающему элементы расписания, назначены слушатели на клик и на долгое удержание. При клике отображается диалоговое окно с заголовком и информации об аудитории. При долгом

удержании появляется меню, через которое можно просмотреть задачи, привязанные к элементу расписания, редактировать этот элемент или удалить его (об этих функциях далее).

6.4.1.1 Создание нового элемента расписания

При нажатии на кнопку «Добавить занятие» происходит запуск модуля создания нового элемента расписания, демонстрируется экран с формой создания нового элемента расписания. Пользователю предлагается ввести информацию о занятии, такую как заголовок, аудитория, а также выбрать из выпадающего списка существующие либо перейти к созданию нового элемента времени, нажав на кнопку «+».

Выпадающий список элементов времени представляет собой элемент типа Spinner с назначенным переопределенным адаптером данных. В нем отображаются отсортированные элементы времени с информацией о временном интервале и именем элемента. В выпадающем списке рядом с каждым элементом имеется кнопка удаления элемента. Удаление происходит только после подтверждения пользователем (посредством диалогового окна) и проверки на то, что данный элемент не используется каким либо элементом расписания, в противном случае отображается соответствующее сообщение.

После подтверждения создания элемента путем клика по соответствующей кнопке происходит проверка корректности введенных данных и в случае успеха новый элемент вносится в базу данных (на самом деле все операции с базой данных происходят не напрямую, а посредством таблицы синхронизации, таким образом помимо сохранения изменений в локальном экземпляре базы данных SQLite происходит также сохранение информации о порядке вносимых изменений, что обеспечивает корректную синхронизацию с облачной базой данных) и становится доступным.

6.4.1.2 Добавление нового элемента времени

Если на экране создания нового элемента расписания пользователь нажал на кнопку создания нового временного элемента, происходит модуля создания нового элемента времени, демонстрируется с формой создания нового элемента времени. Предлагается ввести имя элемента, а также удобным и привычным образом, посредством двух элементов типа TimePicker, указать временной интервал.

После нажатия на кнопку подтверждения следует проверка корректности, и при корректных введенных данных происходит добавление элемента в базу данных и возврат к экрану создания элемента расписания.

6.4.1.3 Редактирование и удаление элемента расписания

Как было сказано ранее, при долгом удержании нажатия по элементу расписания в ListView отображается меню с выбором действия. При нажатии на «Редактировать» запускается модуль редактирования элемента расписания, демонстрируется экран с формой для редактирования элемента расписания, форма аналогична форме для создания нового элемента с тем отличием, что все поля предзаполнены. Пользователь может редактировать элемент и после нажать на кнопку сохранения, произойдет проверка корректности и изменения внесутся в базу данных.

При нажатии на пункт «Удалить» во всплывающем меню отображается диалоговое окно для подтверждения удаления элемента расписания. В случае согласия элемент удаляется из базы.

6.5 Управления задачами для элементов расписания

Задачи для элементов расписания хранятся в таблице со столбцами id, userId, isCompleted, schItemId, text. Первые два столбца нам уже знакомы. Столбец isCompleted хранит логическое значение завершенности задачи,

schItemId хранит идентификатор элемента расписания, к которому относится задача, text собственно отвечает за текст задачи.

6.5.1 На стороне клиента

При долгом удержании нажатии по элементу расписания и клике по пункту меню «Задачи» запускается модуль задач для элементов расписания, демонстрируется экран, отображающий задачи для текущего элемента расписания.

Для отображения списка задач используется элемент типа ListView с переопределенным адаптером данных. Элементы представления задач окрашены либо в зеленый (выполнено), либо в красный (не выполнено) цвет и отсортированы таким образом, что сначала идут невыполненные задачи.

Вверху экрана имеется элемент типа CheckBox, на изменение значения которого установлен слушатель, который запускает метод фильтрации по выполненности задачи у адаптера данных. Это позволяет скрывать/отображать задачи, помеченные как выполненные.

На используемый для отображения задач объект ListView установлен слушатель на долгое удержание нажатия по элементу, который вызывает меню, через которое можно удалить задачу, а также пометить ее как выполненную.

6.5.1.1 Добавление новой задачи

Внизу экрана задач имеется кнопка добавления новой задачи, при нажатии на которую запускается модуль новой задачи и демонстрируется экран с формой для добавления новой задачи. Предлагается ввести текст задачи, и после нажатия на кнопку подтверждения, задача добавляется в базу данных с привязкой к текущему элементу расписания.

6.5.1.2 Просмотр всех задач

На главном экране расписания имеется кнопка «Показать все задания», при нажатии на которую запускается модуль всех задач, демонстрируется экран, отображающий задачи для всех элементов расписания.

Представление этого модуля аналогично представлению модуля, отображающего задачи для текущего элемента расписания, за исключением невозможности добавить новую задачу.

Также помимо сортировки элементов задач по выполненности, присутствует сортировка по дню недели элемента расписания, к которому привязана задача, в зависимости от текущего дня недели. То есть если сегодня среда, то первыми в списке будут невыполненные задачи на четверг, зачем на пятницу и т.д.

6.6 Уровень доступа к таблицам

По умолчанию API-интерфейсы в серверной части мобильного приложения могут быть вызваны анонимно. Далее необходимо ограничить доступ всем клиентам, не прошедшим проверку подлинности. [12]

Установить разрешения доступа к таблицам можно через интерфейс предоставляемый порталом Azure. Для этого в разделе «Простые таблицы» параметром приложения необходимо нажать на «Изменить разрешения» и выбрать для различных операций необходимые уровни доступа. Мы будем использовать «Authenticated access only», что означает доступ только с проверкой подлинности.

Также изменить уровни доступа для каждой таблицы можно, редактируя файл «имя_таблицы.json» в каталоге tables:

```
{
  "softDelete" : true,
```

```
  "autoIncrement": false,
  "insert": {
    "access": "authenticated"
  },
  "update": {
    "access": "authenticated"
  },
  "delete": {
    "access": "authenticated"
  },
  "read": {
    "access": "authenticated"
  },
  "undelete": {
    "access": "authenticated"
  }
}
```

6.6.1 Фильтрация результатов запроса к таблице для разных пользователей

Для ограничения доступа к данным других пользователей, фильтрацию ответа на запрос нужно производить на стороне сервера. Для этого в таблицах имеем столбец userId, отвечающий за хранение идентификатора пользователя, которому принадлежит запись. Также необходимо отредактировать сценарии серверной части Node.js.

```
table.read(
    function (context) {
        context.query.where(function(userId) {
            return this.userId == userId;
        }, context.user.id);
        return context.execute();
    }
);
```

Этот код фильтрует записи в ответе на запрос на чтение, в зависимости от идентификатора авторизованного пользователя. Операции, которые подразумевают выполнение запроса, будут иметь свойство query, в которое можно добавить предложение where. Свойство query представляет собой объект QueryJS, используемый для преобразования запроса OData в то, что может быть обработано серверной частью. [13]

Также необходимо отредактировать сценарий вставки в таблицу, чтобы пользователь не мог подменить поле userId в передаваемом на вставку объекте. Следующий код устанавливает актуальное значение этого поля:

```
table.insert(
  function (context) {
          context.item.userId = context.user.id;
          return context.execute();
});
```

6.7 Информирование пользователя о выполнении запроса к серверу

Реализуем интерфейс ServiceFilter (может быть использован для управления запросами и ответами в HTTP pipeline, используемом MobileServiceClient'ом; ServiceFilter может быть связан с MobileServiceClient через метод WithFilter) в нашем классе ProgressFilter. Он будет отображать представление, переданное ему в конструктор, при выполнении запроса, и по завершению запроса скрывать его. В качестве этого представления мы будем использовать элемент типа ProgressBar, расположенный вверху экрана. Получать объект MobileServiceClient со связанным с ним объектом ProgressFilter будем выше упомянутым методом WithFilter.

Таким образом, при выполнении запроса пользователь будет видеть вращающийся ProgressBar.

6.8 Приведение базы данных к нормальной форме

До этого момента в таблице задач для элементов расписания мы имели следующие столбцы: id, userId, isCompleted, schItemId, text. schItemId является внешним ключом к таблице элементов расписания, в которой также имеется столбец userId. Получаем избыточное хранение.

Мы должны удалить столбец userId из таблицы задач для элементов расписания. Но в тоже время мы должны обеспечить фильтрацию возвращаемых из этой таблицы данных на стороне сервера.

Вопрос бы не стоял, если бы мы обращались к базе данных посредством SQL запросов и использовали операцию JOIN для таблиц и возвращали отфильтрованные по userId результаты. Но мы не можем использовать SQL запросы в серверных сценариях обработки запросов к таблицам. Мы, конечно, можем реализовать кастомное api на стороне сервера. Но тогда, помимо идентичного метода доступа ко всем таблицам через таблицы синхронизации, мы утратим удобный механизм синхронизации данных с облачной базой данных, предоставляемый нам SDK.

Дополнительные трудности создает то, что платформа Azure является относительно новой и активно развивается. Так за время написания данной работы было произведено замещение устаревшей компоненты Azure Mobile Services на Azure Mobile Apps. Естественно, что документация и многие решения для Azure Mobile Services не подходят для Azure Mobile Apps. Официальная документация (датируется в среднем серединой марта 2016) является не полной и описывает только основные моменты.

В ответе службы поддержки к Azure Mobile Services на аналогичный вопрос было предложено использовать представление и обращаться к нему аналогично таблице. Было принято решение попробовать это решение. С помощью Visual Studio было осуществлено подключение к серверу базы данных и создано необходимое представление путем выполнения следующего запроса:

```
CREATE VIEW schTasksWithUserId AS
    SELECT scheduletasks.id, scheduleitems.userid, isCompleted, text, schitemid
    FROM scheduletasks JOIN scheduleitems ON (scheduleitems.id =
                                              scheduletasks.schitemid)
```

Затем путем добавления файлов Json и Javascript была определена соответствующая таблица. Зайдя в портал Azure, автор данной работы увидел в списке таблиц таблицу с именем нашего представления, которая была наполнена корректными данными. Однако при попытке запроса к серверу происходила внутренняя ошибка сервера. Решение оказалось неработоспособным для Azure Mobile Apps.

Было решено в Javascript сценарии обработки запроса чтения, используя функцию объекта таблицы table.read.use, назначить операцию, которая будет выполняться всякий раз перед чтением. В ней мы будем читать данные из таблицы элементов расписания. Затем, используя фильтрацию результатов выполнения запроса чтения к таблице задач на основе полученных из таблицы элементов расписания данных, будем возвращать нужные данные.

Весь код сценария, обрабатывающего чтение из таблицы задач:

```
var queries = require('azure-mobile-apps/src/query');
var schItems;
var readMiddleware = function(req,res,next){
    var table = req.azureMobile.tables('scheduleItems'),
    query = queries.create('scheduleItems');
    table.read(query).then(function(results) {
        if(results){
            schItems = results;
            next();
        }else{
            res.send("no data");
        }
    });
};
```

```
table.read.use(readMiddleware, table.operation);
table.read(
        function (context) {

        var n = schItems.length;
        for (var i = 0; i < n; i++)
        {
            if (schItems[i].userId != context.user.id)
            {
                schItems.splice(i, 1);
                i--;
                n--;
            }
        }

        return context.execute().then(function (results)
        {
            var m = results.length;
            for (var i = 0; i < m; i++)
            {
                var toDel = true;
                for (var j = 0; j < n; j++)
                {
                    if (schItems[j].id == results[i].schItemId)
                    {
                        toDel = false;
                        break;
                    }
                }
                if (toDel)
                {
                    results.splice(i, 1);
                    i--;
                    m--;
                }
```

```
            }
            return results;
        });
    }
);
```

Таким образом, у нас получилось привести базу данных к нормальной форме, обеспечить ограничение возвращаемых данных, и в тоже время сохранить идентичный доступ к таблицам на стороне клиента со всеми преимуществами использования таблиц синхронизации.

6.9 Управление долговременными задачами

Элементы долговременных задач (далее задач) хранятся в таблице со столбцами id, userId, parentId, name, comment, isCompleted, хранящими соответственно: идентификатор элемента расписания, идентификатор пользователя, идентификатор родительского элемента задачи, имя задачи, комментарий к задаче, идентификатор выполненности задачи. Таким образом задачи образуют дерево, что дает возможность для каждой задачи выделять подзадачи. У задач, которые не являются подзадачами, поле parentId помечается строкой «root».

6.9.1 На стороне клиентского приложения

В Android SDK нет отдельного элемента представления для отображения древовидных структур. Для этой цели мы реализуем класс-обертку TaskItemWrap над классом TaskItem, отвечающем за представление элемента таблицы задач, и собственных адаптер данных для элемента представления ListView. Ограничение доступа на стороне сервера происходит аналогично другим таблицам.

В процессе подготовки списка задач назначаются слушатели клика и долгого удержания нажатия по элементу расписания. По клику происходит разворачивание списка подзадач, а в ответ на долгое нажатие демонстрируется меню в котором можно посмотреть подробную информацию о задаче, назначить ее выполненной, добавить напоминание о задаче, добавить подзадачу и удалить задачу.

6.9.1.1 Обертка TaskItemWrap

Наш класс-обертка помимо ссылки на элемент расписания хранит идентификатор свернутости внутренних узлов (позволяет сворачивать/разворачивать подзадачи в списке), уровень вложенности узла, а также список элементов подзадач.

После запуска активности долговременных задач происходит стандартная инициализация локального хранилища и синхронизация данных с сервером. Далее происходит сборка дерева, представляющего собой список из объектов класса TaskItemWrap, у которых нет родительских задач.

Далее на основе уровней вложенности и идентификаторов свернутости происходит построение списка элементов для подачи адаптеру данных.

6.9.1.2 Адаптер данных

Адаптер данных на основе предоставленного списка объектов класса TaskItemWrap строит представление каждой строки списка задач. На основе значений полей класса задачи и класса-обертки задачи устанавливаются необходимые иконки и отступы, а также происходит рекурсивных проход по потомкам для получения информации о числе выполненных и общем числе подзадач для демонстрации в прогрессбаре и в виде текста.

6.10 Уведомления

При выборе пункта меню «Добавить напоминание» про происходит запуск активности, в которой посредством элементов TimePicker и DatePicker происходит сбор информации о времени напоминания. После нажатия на кнопку «добавить» происходит создание объектов Intent и PendingIntent, отвечающих за действие, которое будет выполнено по таймеру. Далее регистрируем наше действие с помощью объекта класса AlarmManager, передавая ему наш PendingIntent и время напоминания.

Для демонстрации оповещения при выполнении действия PendingIntent реализуется класс NotificationCreator, наследуя его от BroadcastReceiver. Главным образом реализуется его метод onReceive, принимающий объект класса Intent хранящий информации о напоминании. В нем происходит конструирование объекта класса Notification и демонстрация напоминания посредством метода класса NotificationManager.

7. Описание пользовательского интерфейса

Далее будут описаны основные элементы, составляющие пользовательский интерфейс.

При первом входе в приложение, либо по истечении срока действия токена авторизации, пользователю предлагается авторизоваться через учетную запись Google.

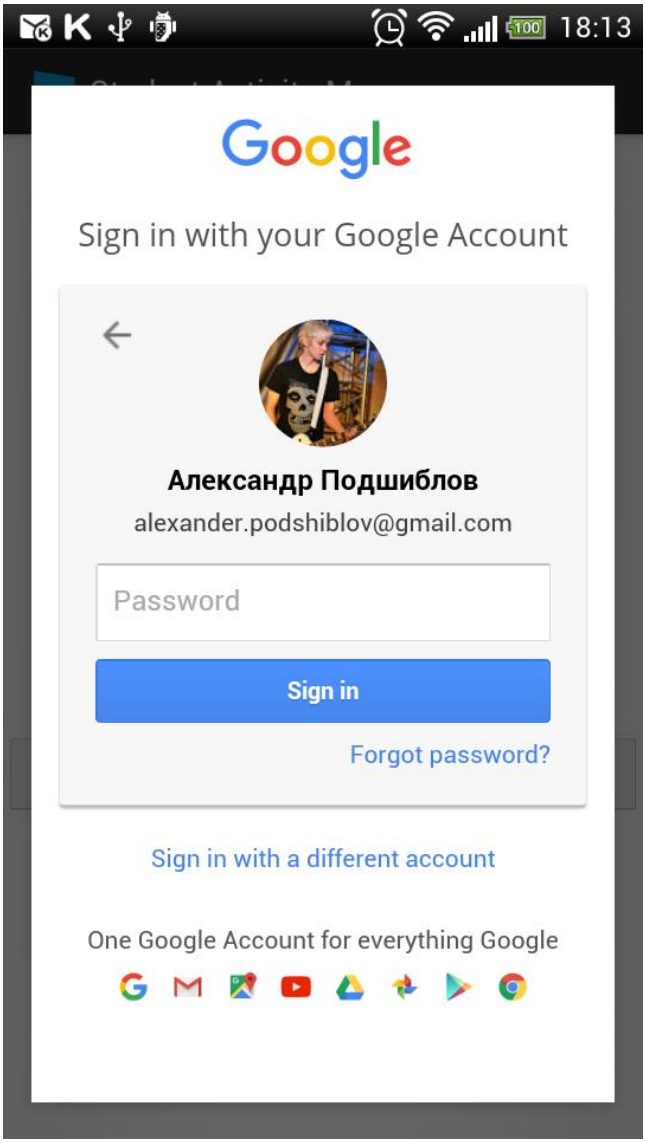

Рис. 5. Экран авторизации пользователя

После прохождения авторизации, если пользователь не зарегистрирован, ему предлагается пройти регистрацию, путем ввода данных об учебном заведении и факультете. При начале ввода пользователю демонстрируется список из уже существующих в базе учебных заведений и факультетов.

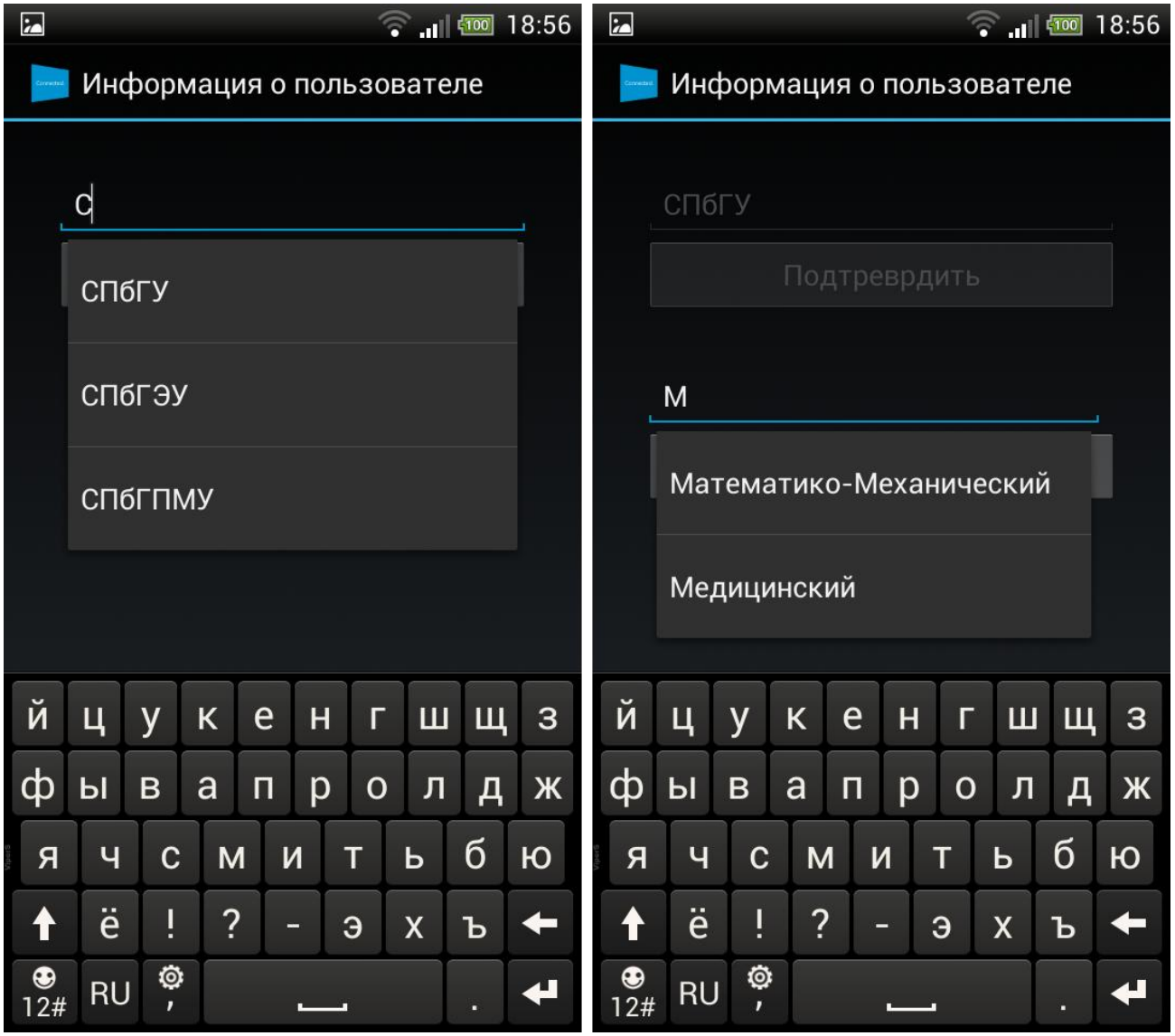

Рис. 6. Экраны регистрации пользователей

После прохождения регистрации или после подтверждения зарегистрированности пользователь видит стартовый экран с кнопками перехода к экрану расписания и экрану долговременных задач. Во время всех попыток связаться с сервером (при загрузке базы существующих учебных заведений или факультетов, во время проверки регистрации и во врем синхронизации данных с сервером) пользователю демонстрируется вращающийся прогрессбар. Так после перехода к экрану расписания происходит попытка синхронизации и пользователь видит прогрессбар.

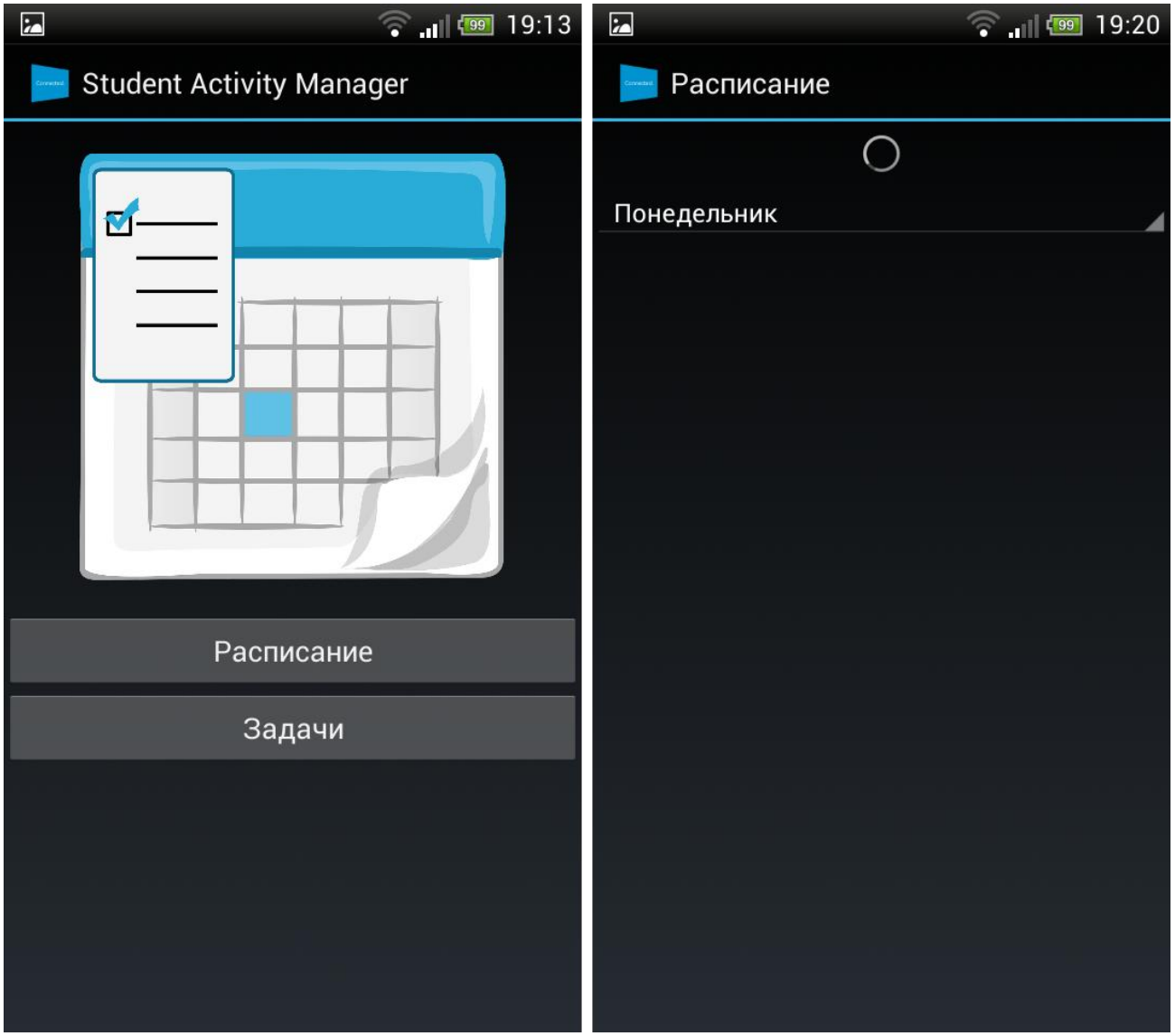

Рис. 7 Стартовый экран и экран расписания во время синхронизации

После загрузки, прогрессбар изчезает и расписание становится доступным. Вверху экрана находится элемент, позволяющий переключаться между днями недели. После списка с элементами расписания находятся кнопки добавления нового элемента в расписание и кнопка перехода к экрану со всеми задачами для элементов расписания. При удержании нажатия по элементу расписания демонстрируется меню, позволяющее перейти к задачам для этого элемента расписания, к редактирования элемента и к его удалению.

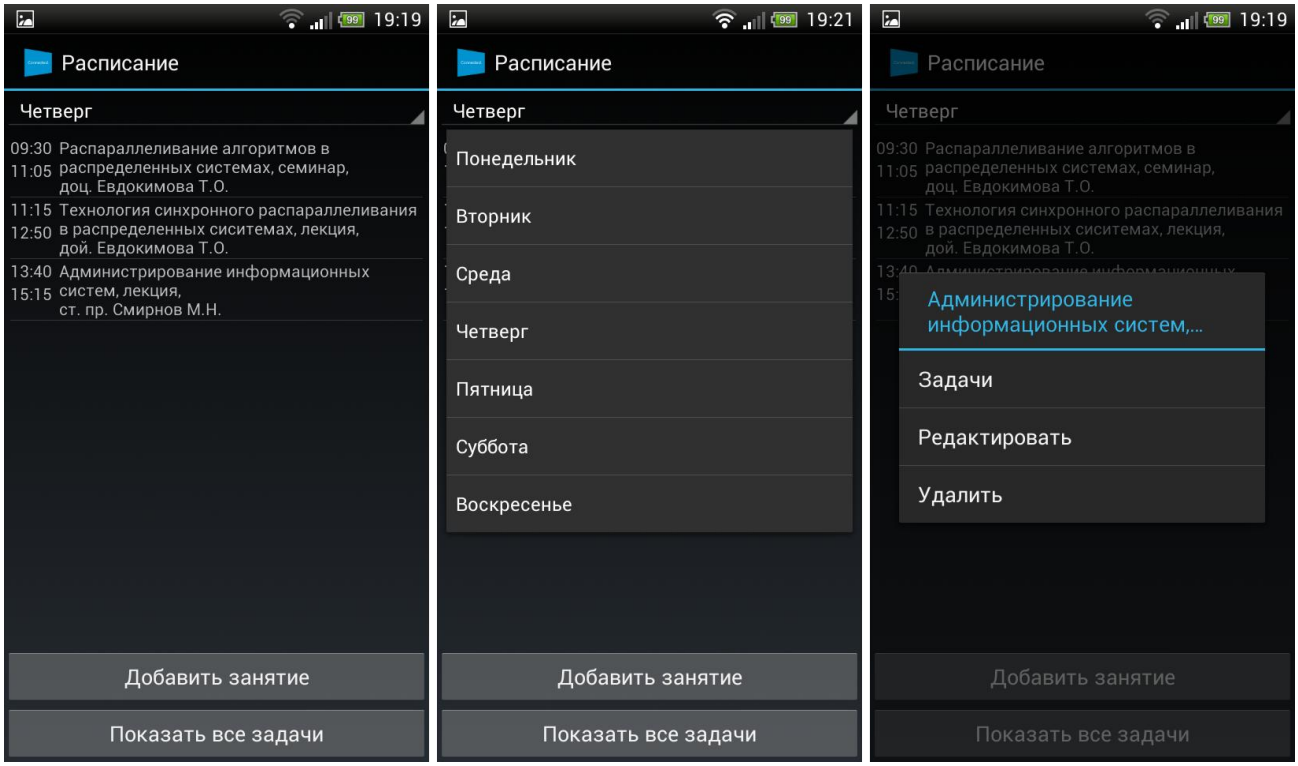

Рис. 8. Экран расписания

При переходе к созданию нового элемента расписания демонстрируется экран добавления этого элемента. На нем расположены поля для ввода заголовка, аудитории, а также выпадающий список выбора элемента времени, на котором также расположены кнопки удаления соответствующих элементов. Ниже расположены кнопка добавления нового элемента времени и кнопка добавления элемента расписания.

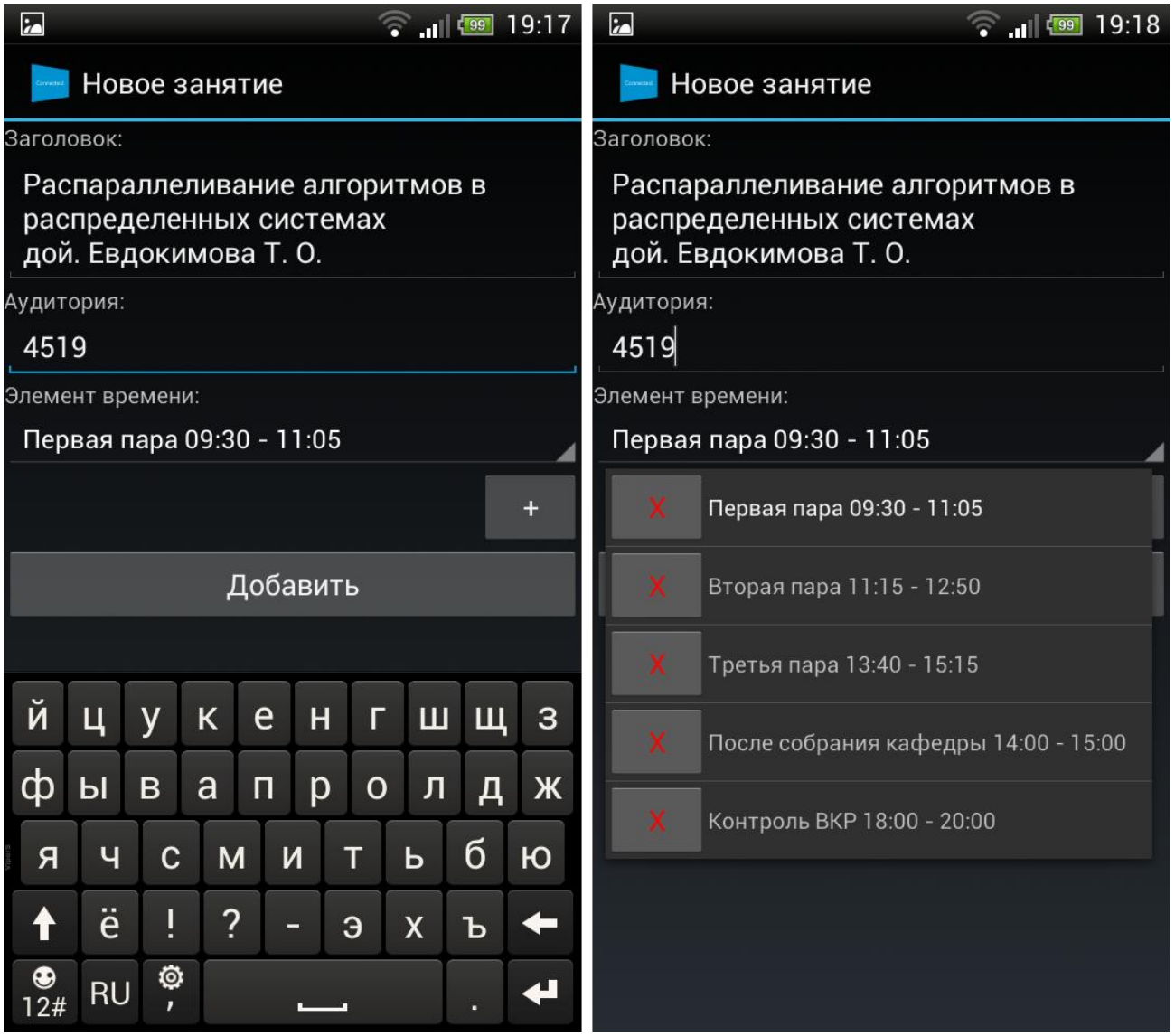

Рис. 9 Экран добавления нового элемента расписания

По нажатию на кнопку добавления нового элемента времени происходит переход к экрану добавления нового элемента времени. На нем расположены поле ввода названия элемента, а также два элемента для установки времени начала и конца временного интервала. Ниже расположения кнопка добавления элемента времени.

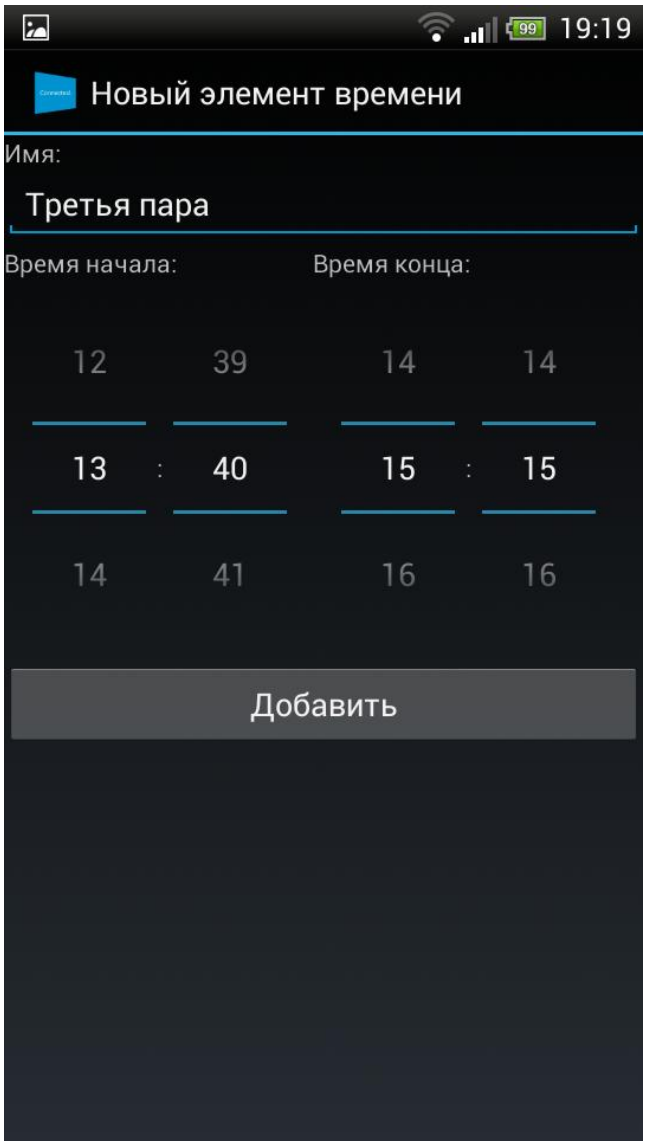

Рис. 10. Экран добавления элемента времени

При нажатии на пункт «Задачи» в меню, появляющемся при удержании нажатия по элементу расписания, демонстрируется экран задач для текущего элемента расписания. На нем распложены задачи, окрашенные в соответствующий цвет в зависимости от их выполненности. Имеется чекбокс, позволяющий скрыть выполненные задачи, а также кнопка добавления новой. При удержании нажатия по задаче появляется меню, позволяющее отметить задачу как выполненную или удалить ее.

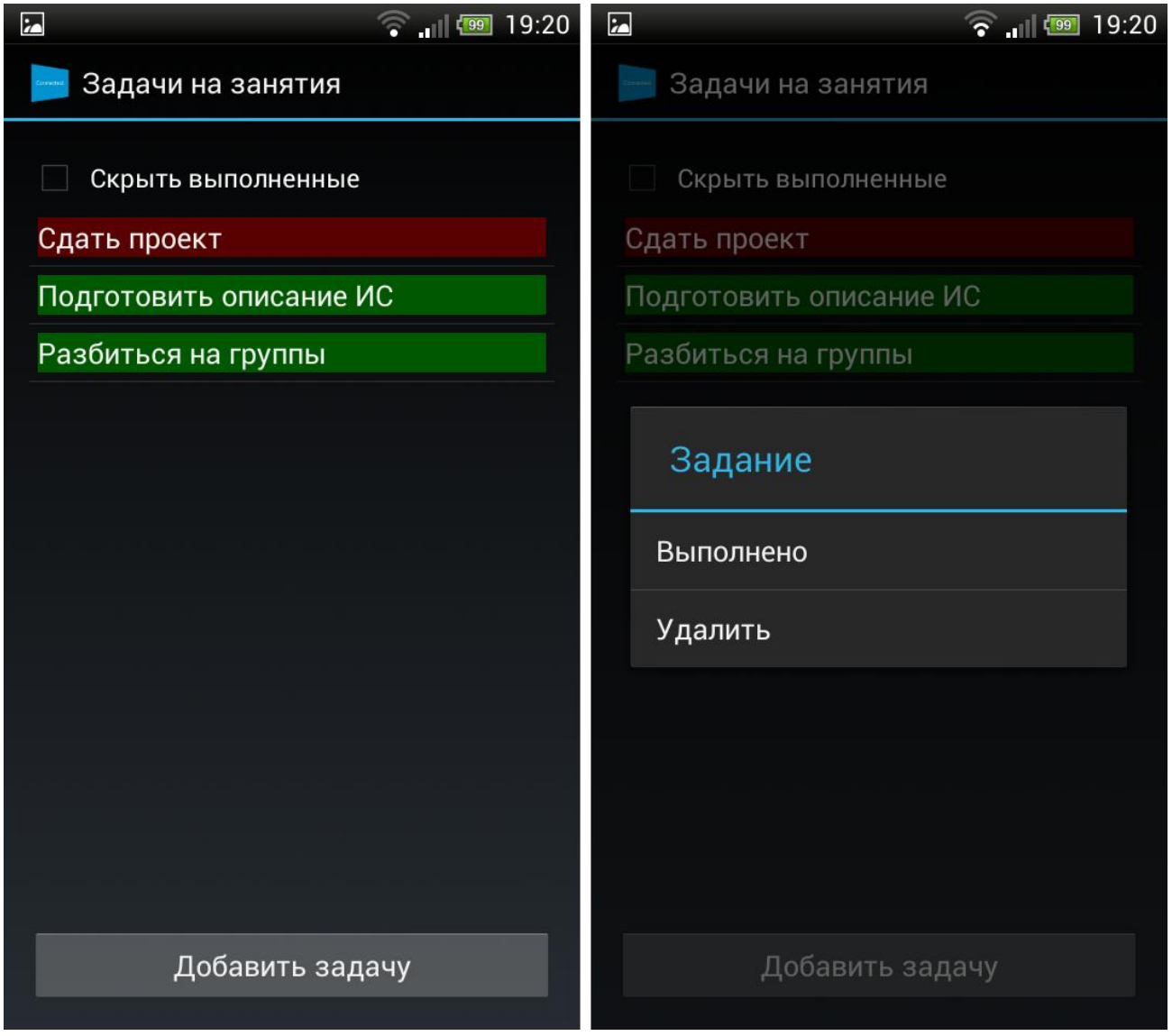

Рис. 11. Экран задач по текущему элементу расписания

Нажатие на кнопку «Все задачи» экрана расписания приводит к демонстрации экрана задач для всех элементов расписания. Он аналогичен экрану задач по текущему элементу расписания. Отличие состоит в невозможности из этого экрана добавить новую задачу, а также в том, что здесь задачи сортируются в зависимости от дня недели элемента расписания, к которому они привязаны, и текущего дня.

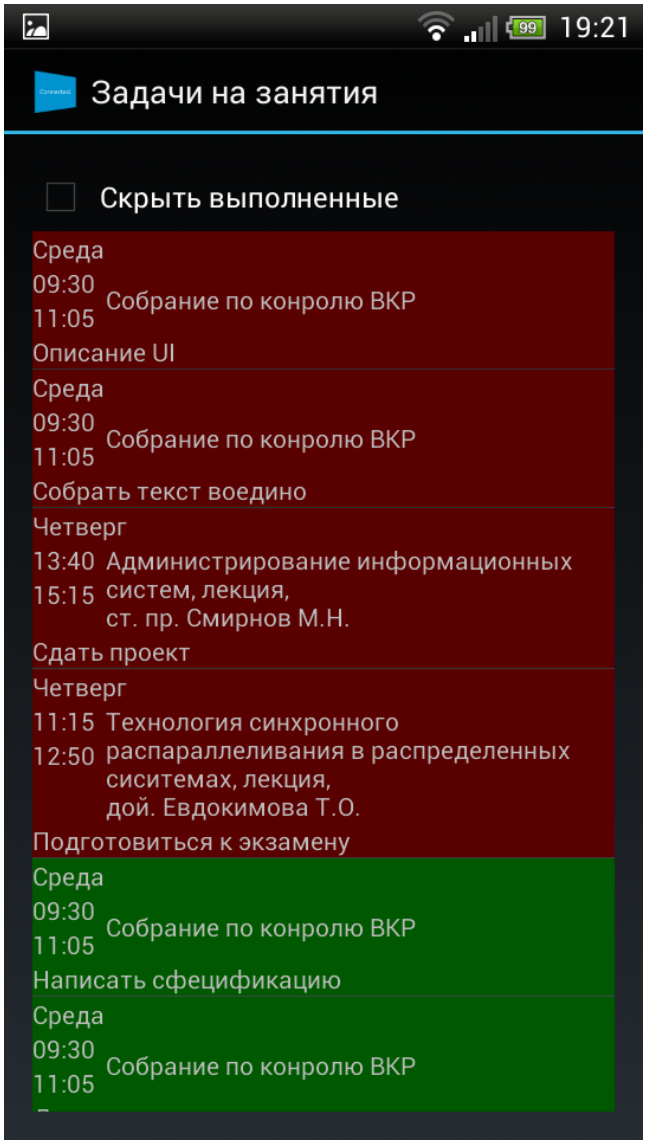

Рис. 12. Экран задач для всех элементов расписанием

После перехода к экрану долговременных задач происходит попытка синхронизации с сервером, во время которой демонстрируется вращающийся прогрессбар. На данном экране демонстрируется древовидная структура задач. При нажании на задачу разворачивается список подзадач. Выполненные задачи помечаются голочкой, иначе – восклицательным знаком. Прогрессбар указывает на прогресс выполнения задачи на основе выполненности потомнов. Над ним цифрами указано число выпоненных непосредственных потомков / общее их число (в скобках всех потомков). Внизу экрана расположена кнопка добавления задачи в список.

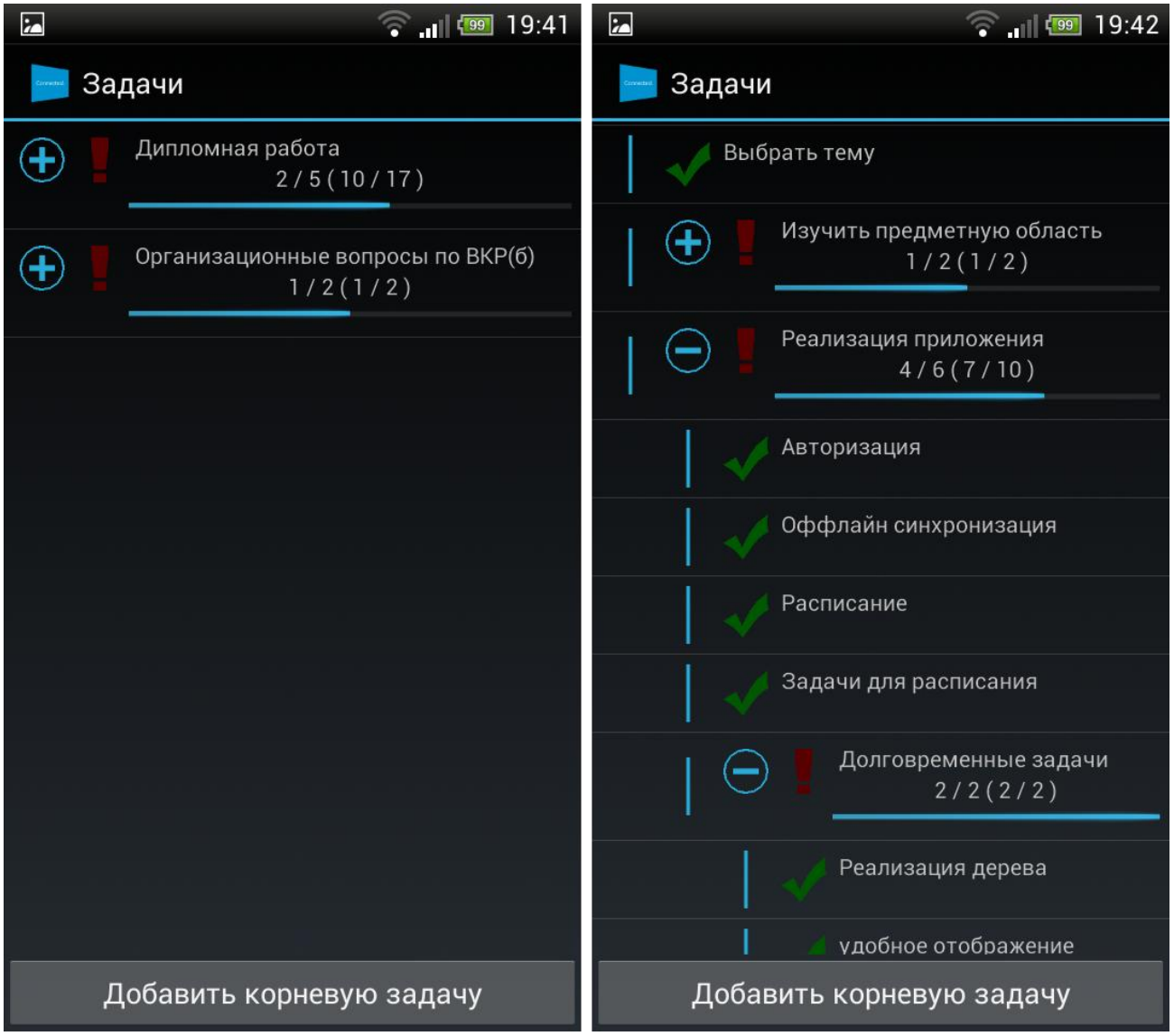

Рис. 13. Экран долговременных задач

При долгом удержании нажатия на задачу в списке демонстрируется меню, позволяющее выполнять различные действия для элемента задачи. При нажатии на пункт «Подробнее» демонстрируется диалоговое окно с подробной информацией о задаче.

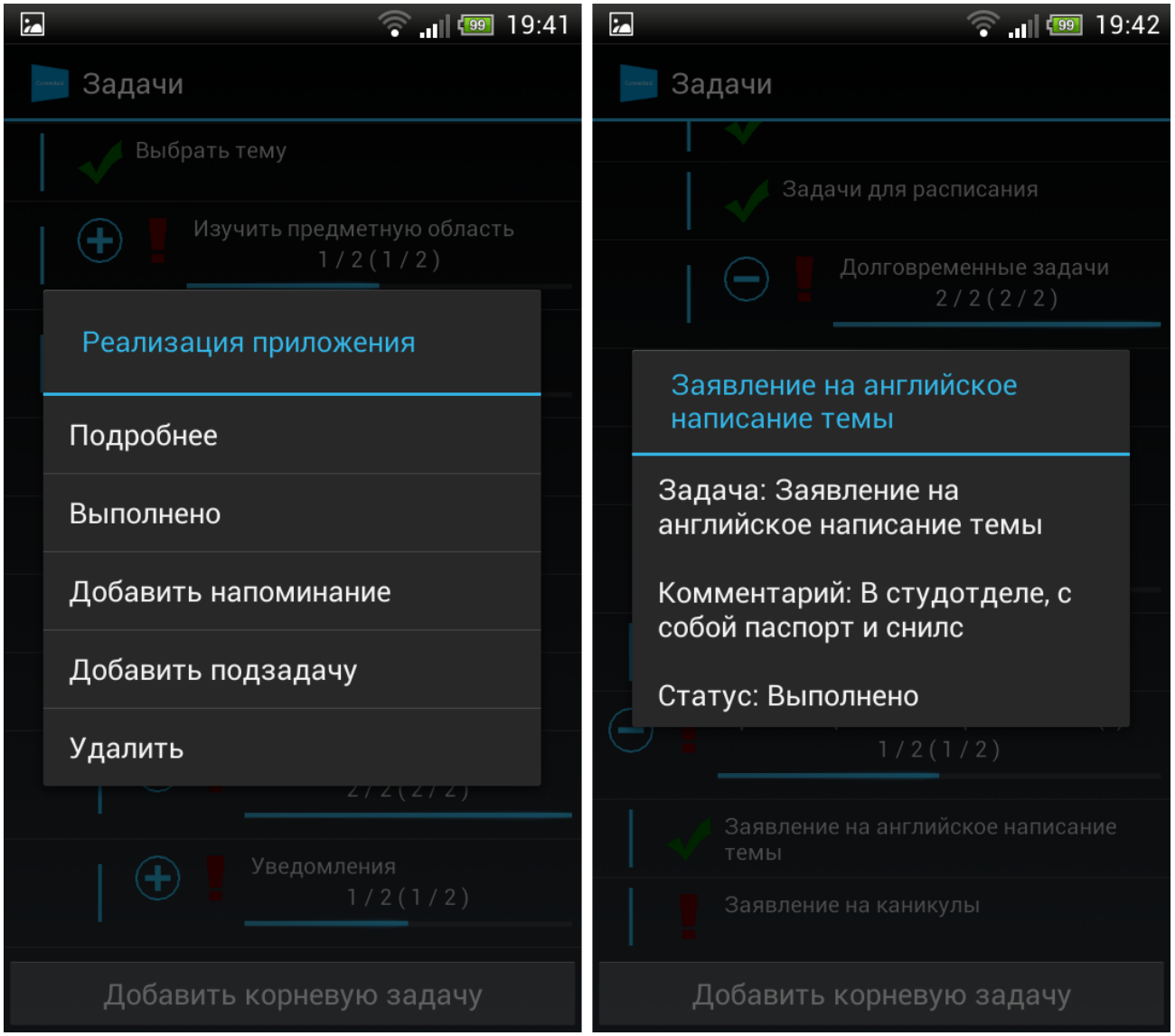

Рис. 14. Меню элемента задач и окно подробной информации о задаче

При нажатии на пункт меню «Добавить напоминание» предлагается ввести дату и время напоминания и при нажатии на кнопку «Сохранить» напоминание регистрируется и в будущем демонстрируется пользователю.

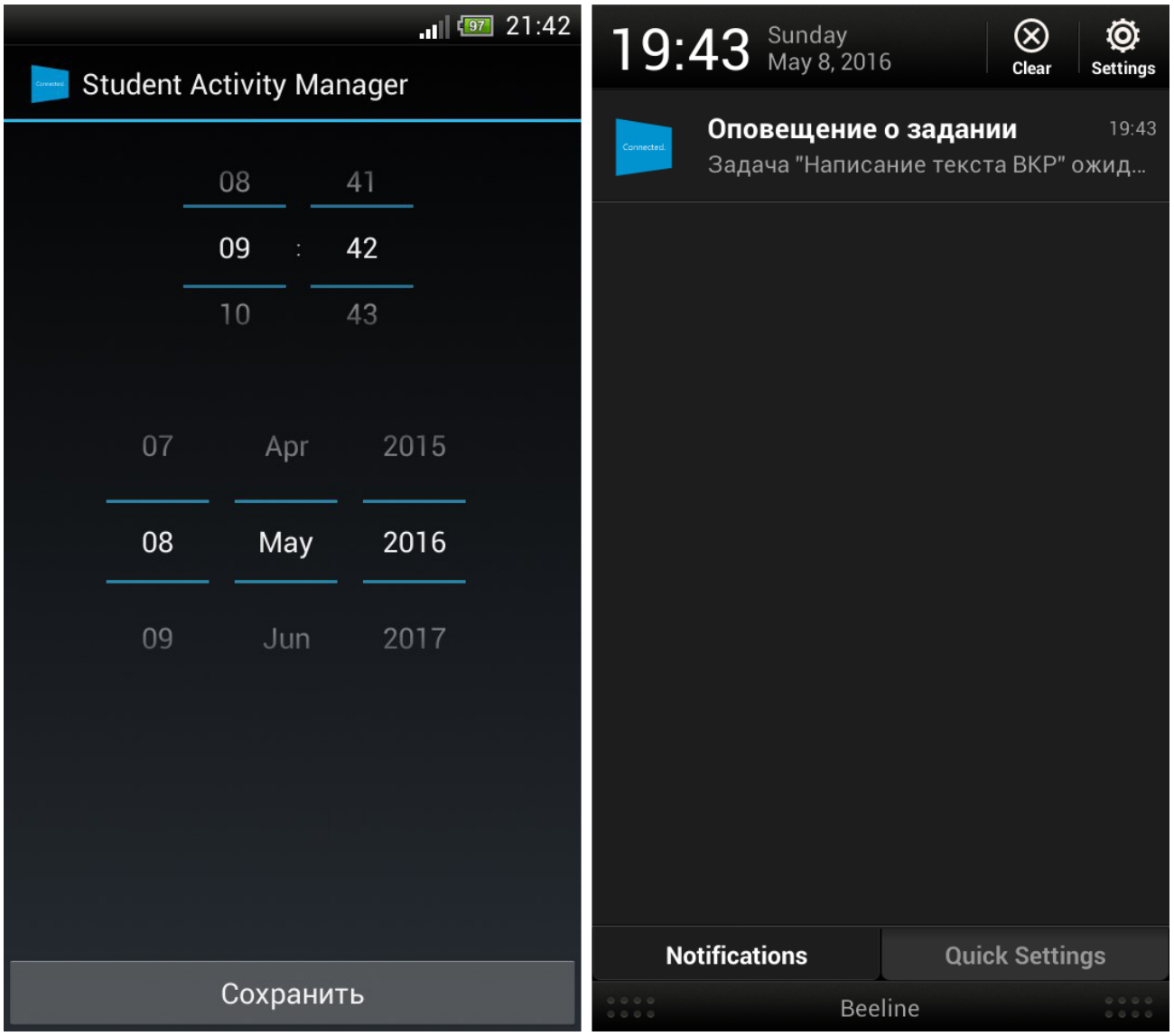

Рис. 15. Экран добавления напоминания и оповещение в статусбаре

8. Заключение

8.1 Результаты

В результате работы разработано и реализовано приложение, которое позволяет пользователю вести расписание учебных занятий, учет поставленных задач и быть уверенным в сохранности своих данных. Исходный код приложения клиента[14] и сервера[15] доступен в публичном репозитории на GitHub.

Приложение позволяет:

- Выполнять авторизацию и аутентификацию с помощью учетной записи Google
- Управлять расписанием
- Управлять задачами для элементов расписания
- Управлять древовидной структурой долговременных задач
- Назначать и получать напоминания
- Хранить данные в облачной структуре
- Работать автономно

Далее указаны использованные в процессе реализации приложения языки программирования и разметки, а также соответствующее число строк кода:

- Java – 3264
- XML – 693
- Javascript –263

Приложение удовлетворяет требованиям и критериям, предъявленным к нему на этапе постановки задачи. К сожалению, не удалось реализовать весь планируемый функционал, такой как: полная функциональность уведомлений от приложения о необходимости выполнения той или иной задачи. Это связано со следующими факторами:

- Ограниченность автора по времени

- Новизна платформы и как следствие частые обновления
- Замена компоненты Azure Mobile Services на Azure Mobile Apps
- Недостаточная документация новой платформы
- Большие временные затраты на изучение платформы

8.2 Дальнейшая работа

Дальнейшая работа над приложением возможна в следующих направлениях:

- Разработка алгоритма анализа загруженности пользователя
- Добавления напоминай о необходимости выполнения долгосрочной задачи на основе анализа загруженности

Список литературы

[1] Органайзер iStodo - кибернетический помощник студента – Электронные текстовые данные. – 2015. Режим доступа: http://istodo.ru/, свободный.

[2] Приложения на Google Play – Timetable – Электронные текстовые данные. – 2015. Режим доступа: https://play.google.com/store/apps/details?id=com.gabrielittner.timetable, свободный.

[3] Сафонов В.О. Развитие платформы облачных вычислений Microsoft Windows Azure : онлайн-курс лекций. – ИНТУИТ, 2013

[4] Доминик Беттс, Алекс Гомер, Алехандро Езерски, Масаши Нарумото, Ганц Чжан. Перенос приложений в облако на базе платформы Microsoft Windows Azure – 2012. – 179стр.

[5] Официальная страница проекта Windows Azure на сайте компании Microsoft / Компания Microsoft – Электронные текстовые данные. – 2015. Режим доступа: http://azure.microsoft.com/ru-ru/, свободный.

[6] Мобильные приложения. Создание привлекательных приложений для iOS, Android и Windows / Компания Microsoft. – Электронные текстовые данные. – 2015. Режим доступа: http://azure.microsoft.com/ruru/services/app-service/mobile/, свободный.

[7] Проверка подлинности и авторизация в мобильных приложениях Azure. Электронные текстовые данные. – 2016. Режим доступа: https://azure.microsoft.com/ru-ru/documentation/articles/app-service-mobile-auth/ , свободный.

[8] Использование пакета SDK Node.js для мобильных приложений Azure. Обзор операций с таблицами. Электронные текстовые данные. – 2016. Режим доступа: https://azure.microsoft.com/ru-ru/documentation/articles/app-service-mobile-node-backend-how-to-use-server-sdk/#TableOperations, свободный.

[9] Использование пакета SDK Node.js для мобильных приложений Azure. Практическое руководство. Изменение кода в Visual Studio Team Services. Электронные текстовые данные. – 2016. Режим доступа: https://azure.microsoft.com/ru-ru/documentation/articles/app-service-mobile-node-backend-how-to-use-server-sdk/#online-editor , свободный.

[10] Использование клиентской библиотеки Android для мобильных приложений. Определение клиентских классов данных. Электронные текстовые данные. – 2016. Режим доступа: https://azure.microsoft.com/ru-ru/documentation/articles/app-service-mobile-android-how-to-use-client-library/#-2, свободный.

[11] Электронные текстовые данные. – 2016. Режим доступа: https://azure.microsoft.com/ru-ru/documentation/articles/app-service-mobile-android-get-started-offline-data/, свободный.

[12] Включение автономной синхронизации для мобильного приложения Android. Электронные текстовые данные. – 2016. Режим доступа: https://azure.microsoft.com/ru-ru/documentation/articles/app-service-mobile-android-get-started-users/, свободный.

[13] Использование пакета SDK Node.js для мобильных приложений Azure. Практическое руководство. Обязательная проверка подлинности для доступа к таблицам. Электронные текстовые данные. – 2016. Режим доступа: https://azure.microsoft.com/ru-ru/documentation/articles/app-service-mobile-node-backend-how-to-use-server-sdk/#howto-tables-auth, свободный.

[14] Исходные коды системы. Клиент / Alexander Podshiblov. Электронные текстовые данные. – 2016. – Режим доступа: https://github.com/alexander-podshiblov/Student-Activity-Manager, свободный.

[15] Исходные коды системы. Сервер / Alexander Podshiblov. Электронные текстовые данные. – 2016. – Режим доступа: https://github.com/alexander-podshiblov/SAM---Server, свободный.